2015

Compressive Sensing Analog Front End Design In 180 nm CMOS Technology

Julin Mukeshkumar Shah
Wright State University

Compressive Sensing Analog Front End Design in 180 nm CMOS Technology

A thesis submitted in partial fulfillment
Of the requirements for the degree of
Master of Science in Engineering

By
Julin M Shah
B.E., Gujarat Technological University, 2012

2015
Wright State University

WRIGHT STATE UNIVERSITY
GRADUATE SCHOOL

August 18, 2015

I HEREBY RECOMMEND THAT THE THESIS PREPARED UNDER MY SUPERVISION BY Julin Mukeshkumar Shah ENTITLED Compressive Sensing Analog Front End Design in 180 nm CMOS Technology BE ACCEPTED IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE DEGREE OF Master of Science in Engineering.

_____
Chien-In Henry Chen, Ph.D.
Thesis Director

_____
Brian D. Rigling, Ph.D.
Chair, Department of Electrical Engineering
College of Engineering and Computer Science

Committee on Final Examination

_____
Chien-In Henry Chen, Ph.D.

_____
Marian K. Kazimierczuk, Ph.D.

_____
Jiafeng Xie, Ph.D

_____
Robert E. W. Fyffe, Ph.D.
Vice President for Research and
Dean of the Graduate School

ABSTRACT

Julin M Shah. M.S.Egr., Department of Electrical Engineering, Wright State University, 2015.
Analog Front End Design of the Compressed Sensing in TSMC 180 nm CMOS Technology.

In order to accurately reconstruct signal waveform a signal must be sampled at least twice as fast as the bandwidth of the signal. Ultra Wideband (UWB) signals have extraordinary potential for high information transmission while a central focus of wireless has been the mobile communication. It is an emerging area that involves development of RF sensing and spectral applications over multiple GHz bandwidths. Even though our technology is improving, it is very challenging to build ADC's that are compatible and keep up with the growth of ultra-wideband range. Compressive sensing does "sampling" and "compressing" at the same time and exploits the sparsity for commensurate power saving by enabling sub-Nyquist under-sampling acquisition. The main idea behind compressive sensing is to recover specific signals from very few samples as compared to conventional Nyquist samples. In this thesis, a compressive sensing front-end (CSFE) is designed and analyzed to mitigate sampling approach limitations of the architecture in a CMOS process. CSFE has four main components: pseudo random sequence generator (PBRS), multiplier, integrator, and ADC. The PBRS (implemented by a Gold code generator) and the multiplier are designed in Cadence Spectre using TSMC 180nm technology. The integrator and the 10-bit ADC are designed and verified using both Verilog-A and Matlab. Using 4 GHz PBRS and 800 MHz under sampling ADC, the CSFE design can detect signal frequency up to 2 GHz after applying the Orthogonal Matching Pursuit algorithm to reconstruct the under-sampling ADC data.

TABLE OF CONTENTS Page

LIST OF FIGURES Page

LIST OF TABLES Page

Acknowledgement

I would like to express my gratitude to my advisor Dr. Henry Chen for his continuous support, motivation and guidance. I would like to thank my committee members (Drs. Marian K. Kazimierczuk and Jiafeng Xie) for their kindness and time. I am also thankful to VLSI professors for sharing their knowledge. I would also like to thank Ethan Lin for sharing his views and for spending his time enlarging the scope of my work. I am thankful to my teachers for their support.

Chapter 1
Introduction

1.1 Conventional technique for sampling

In order to reconstruct a band limited signal we use the Nyquist/Shannon theorem which states that, the sampling rate should be at least two times of a signals bandwidth. Assume our signal in a time domain is x (t), which is a continuous time signal and X (f) is a Fourier transform of x (t). We can represent X (f) as

$$X(f) = \int_{-\infty}^{\infty} x\,(t) e^{-i2\Pi ft}\, dt \qquad (1)$$

Signal x (t) can be considered as band-limited if we have some bandwidth B such that X(f) is zero for all the values other than |f|>B, according to the Nyquist theorem to reconstruct the original signal successfully we have to take samples at a rate which is twice that of the bandwidth of the original signal [15].

1.2 Limitation of Nyquist theorem

We are into the world of "BIG DATA" and its growing fast. We are using ultra wideband to transmit signals that are higher than 500 MHz. According to the

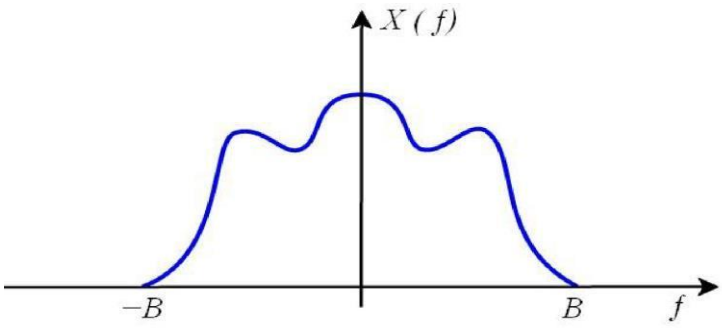


Figure 1.1 Magnitude of the Fourier transform of a bandlimited function [15]

Nyquist theorem, hardware should be capable of working in Gigahertz (GHz) range. Even though our technology is improving, our technology is not capable of fulfilling the requirement of current applications. The main component in the hardware to sample data is an ADC (Analog – Digital Converter) and it is very challenging to build a high-resolution ADC that is compatible in the ultra-wideband range.

1.3 Overview of Compressed Sensing

Compressed sensing was firstly introduced by Candes, Tao, Donoho and Romberg in 2006. Compressed sensing also known as compressive sensing (CS) is a technique, which does "sampling" and "compressing" at the same time. The main idea behind compressed sensing is to recover specific signals from very few samples as compared to traditional Nyquist samples. Compressed Sensing relies on two key principles that are Sparsity and Incoherence [5].

Recent research in the field of signal processing states that we can transform most of the signals into other forms in which that signal has sparse representation. Sparse signal refers to a signal in which most of the information is contained by very few number of its component, all the other components in that signal contains very few or no information and you can discard that information. The signal can be transformed back to its original form by keeping those few information and their locations. Example of a sparse signal is shown in the image below where on the left is the original image, at the center is its wavelet transform and the image on the right is the image which is recovered from the wavelet transform. In wavelet transform you can see that very few coefficients contains large amount of data and the rest has very little to no data. The image on

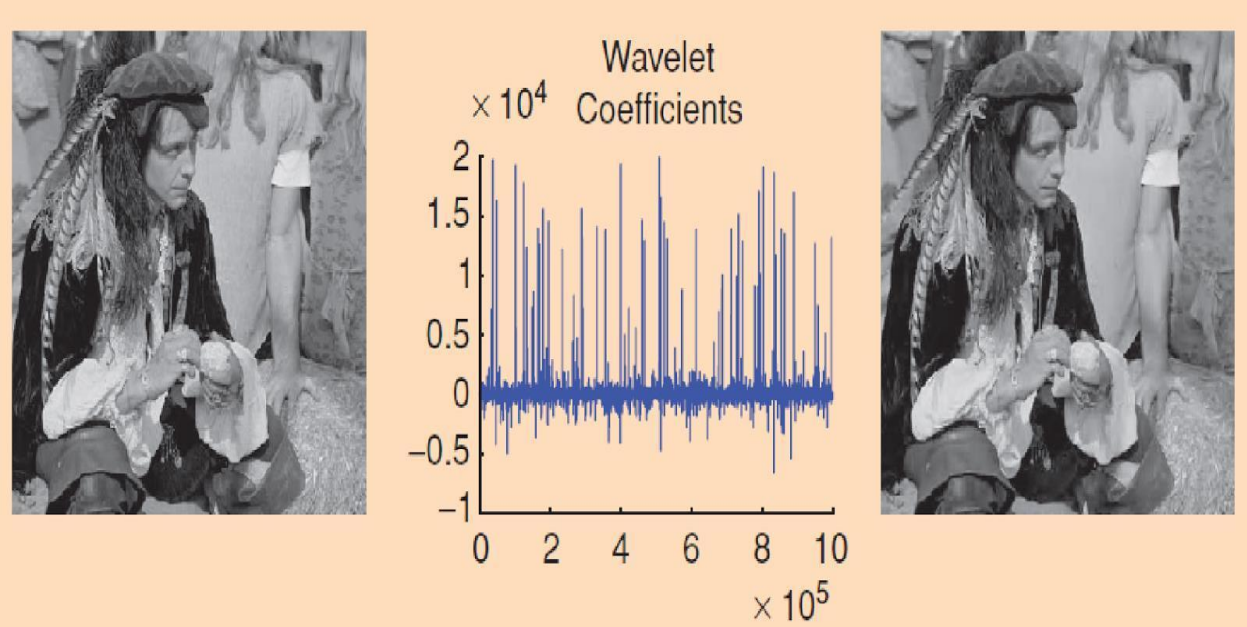


Figure 1.2 Compressed Sensing example [5]

The right is constructed from only 2.5% of the larger wavelet coefficients and the quality of the image is not that degraded.

1.4 Problem Statement

Compressed sensing is a signal processing technique that exploits this sparsity for commensurate power saving by enabling sub-Nyquist under-sampling acquisition. In this study CMOS compressive sensing front-end design will be designed and analyzed to mitigate sampling approach limitations of the conventional architecture. All the front-end components are designed in TSMC 180nm CMOS technology.

1.5 Organization of Thesis

Chapter-2 discusses compressed sensing operation, basic mathematical models, and important theorems which supports compressed sensing. Brief discussion about previous analog front-end design is also provided. Chapter 3 presents proposed design of random demodulator. Schematic design and simulation results of Gold code generator, 1-bit multiplier, Integrator and under-sampling ADC are discussed. Chapter 4 discusses performance evaluation of compressed sensing front-end design. In chapter 5 we concluded the compressed sensing front-end design, its impact and contribution of it. We also discuss future work to improve our design limitations.

Chapter 2

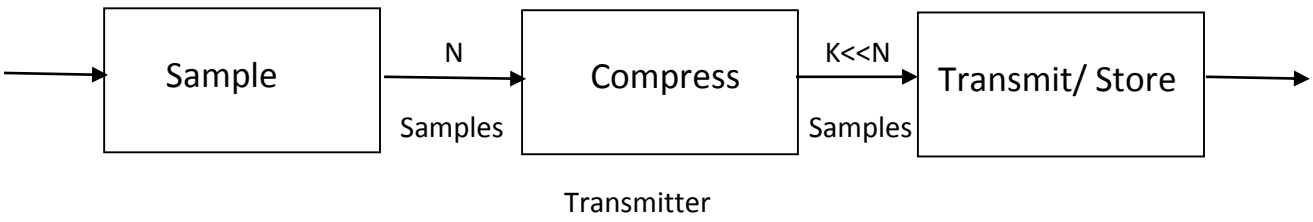
Compressed Sensing

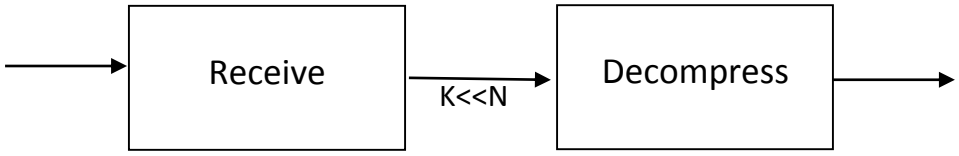
2.1 Compressed sensing problem

Let's start with the input signal $X \in R^{N \times 1}$, which is a real valued, finite and discrete one dimensional time signal, in which we consider signal X as a $N \times 1$ column vector. We can represent any picture or any data by vectorising it in an one-dimensional matrix. Any signal in the R^N can be treated as a basis of $N \times 1$ vectors $\{\Psi_i\}_{i=1}^{N}$ and we consider basis as an orthonormal. Any discrete signal x can be represented as

$$X = \sum_{n=1}^{N} \alpha_n \Psi_n = \Psi\alpha \quad (1)$$

$\alpha = [\alpha_1, \alpha_2, \alpha_3, \alpha_4 \ldots\ldots\ldots \alpha_n]$ is a transform coefficient vector that is calculated as $\alpha_n = (X, \Psi_n)$, here we can say that x and α both are representing the same signal but the difference is x represents it in time domain and α represents it in Ψ domain [13]. Here we can define this signal as K-sparse if we have K coefficients with large amplitude and all other N-K coefficients with zero or very small amplitude. In this case we are interested in K<< N. The signals that we are considering must be sparse . Let's consider an example of digital camera:





Receiver

Figure 2.1 Conventional Compression paradigms

In a digital camera whenever you capture an image it will take N samples of that image and after that when you save that image using JPEG format, it will represent it in wavelet transform in which there are K coefficients that have large amplitude and all other N-K have zero or small amplitude. So basically what we do here is taking all the N samples and after that ignoring N-K samples from it and proceed with K samples. The main drawback of this is, first we have to start with large number of N samples and we have to store those N samples. Second, we have to compute all the N coefficients even though we are going to discard N-K of them.

Compressed sensing can overcome these drawbacks by directly getting compressed signal, it directly acquires K samples rather than getting N samples and then discarding N-K samples. Compressed sensing is a linear measurement process that processes M < N samples which are obtained by multiplying X and measurement matrix Θ (M X N) [13]. Here you can write Y as

$$Y = \Phi X = \Phi\Psi\alpha = \Theta\alpha \qquad (2)$$

Here y =Θα, Θ is $M \times N$ matrix.

Stable measurement matrix can be designed in CS such that it converts any signal X into K-sparse signal and does not damage any information from signal X and also we have to find the

reconstruction algorithm that can recover signal x or similar to signal X from K-sparse signal.

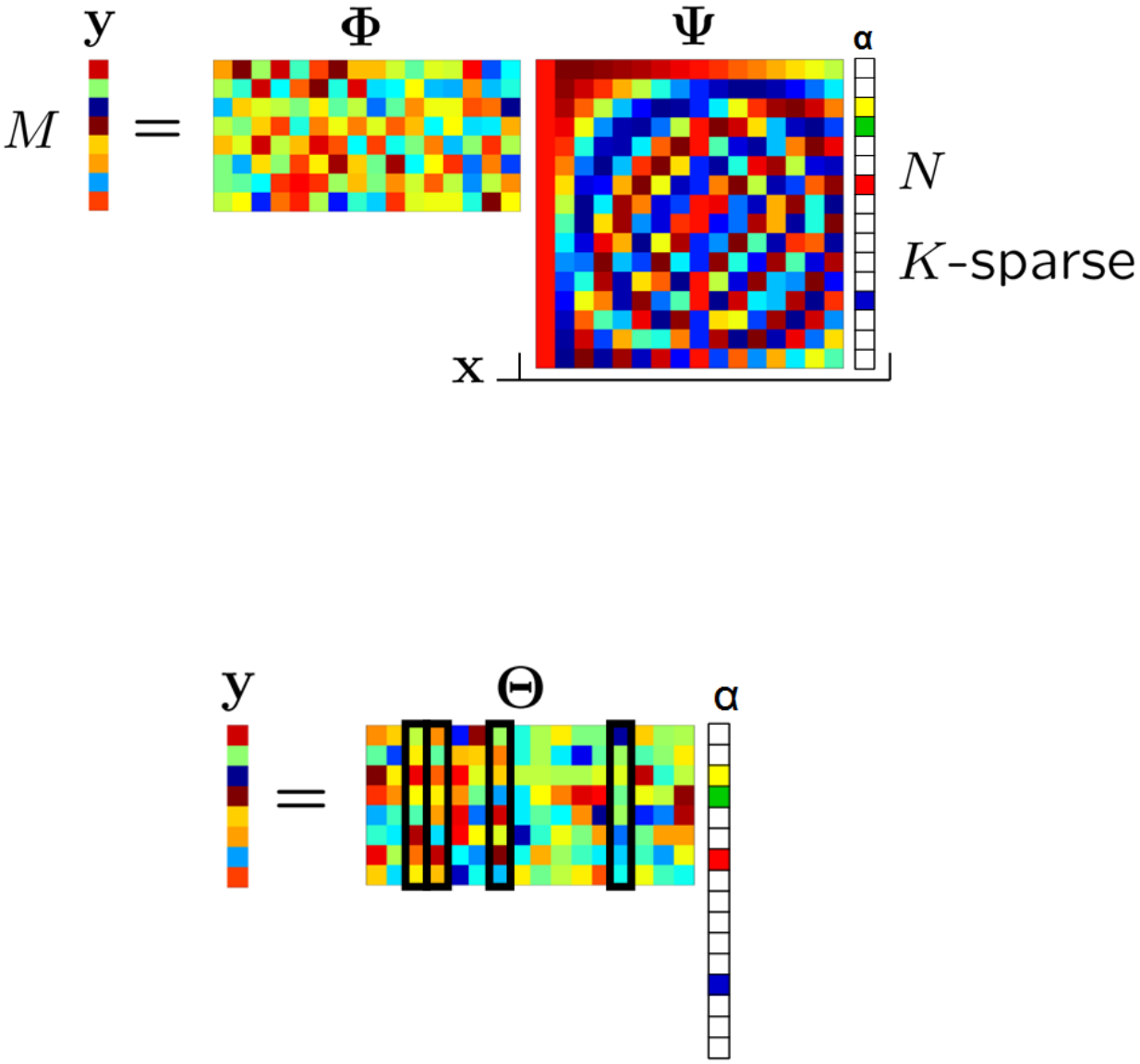


Figure 2.2 Matrix representation [13]

2.2 Outlining Measurement matrix

In compressed sensing, our first task is to design a linear measurement matrix Θ such that we can encode signal X (N × 1 matrix) to the measurement Y (M × 1 matrix) without losing any information of X and we should be able to recover all the information back from Y. To recover N × 1 measurement from the M × 1 (M < N) measurement is ill-poised problem because we have less number of equation than the number of unknown variables. Compressed sensing is all about sparse signals so if our signal is K-sparse(K < M) then measurement Y is a multiplication of K nonzero columns of Θ and α, if we know all the location of the K nonzero entries then M × K system of linear equation can be formed to get all the nonzero entries[13]. The most important condition to get the data back, for any other signal V which has the same K nonzero entries is,

$$1 - \epsilon \leq \frac{\|\Theta V\|_2}{\|V\|_2} \leq 1 + \epsilon \tag{3}$$

There is one another way to ensure compressed sensing and that incoherence which states that, linear measurement matrix Θ must be incoherent with the signal in Ψ basis or you can say that Θ cannot sparsely represent signal in Ψ basis. So our task is to design a matrix Θ such that y = Θα, satisfy the RIP property. In compressed sensing we can take Θ as a random measurement matrix and elements of this matrix will be independent and identically distributed (IID) then the output vector with M × 1 measurements will be the linear representation of N × 1 measurement of signal X. So the two main properties to recover the data is Θ is incoherent with basis $\Psi = I$ of delta spikes with high probability, and other is M × N independent and identically dependent matrix $\Theta I = \Theta$ must satisfy RIP if $M > cK \log \frac{N}{K}$ where c is a small constant. So now we can take any random matrix with ±1 entries and zero mean as Θ and it will satisfy RIP and universal properties.

2.3 Reconstruction algorithms:

In compressed sensing, once we mapped all the information of K-sparse signal X to the compressed M × 1 measurement of Y, we have to find a way to reconstruct N × 1 signal X or similar to signal X from the y measurements. We have M measurements which is less than N, so for any α' there can be infinite solutions of Θα' = Y. For example, for any vector r there can be Θα = Y and also Θ (α + v) = Y. So our aim for reconstruction is to find α similar to the original signal. We can define ℓp norm of vector α as$(\|\alpha\|_p)^p = \sum_{i=1}^{N}|\alpha|_p$, in this equation if we give p as 0 then we will get norm as 0 [13].

ℓ_2 minimization approach: Let's consider the **ℓ_2** norm which is a classical approach for solving inverse problems, inverse vector X or similar to X can be

$$\widehat{X} = \text{argmin } \|X'\|_2 \text{ such that } \Theta X' = Y \tag{4}$$

But unfortunately the vector which we get using **ℓ_2** norm is non sparse with lots of ringing so we cannot use this algorithm for compressed sensing [13].

ℓ_0 minimization approach: Another way to find the solution of inverse sparse vector is **ℓ_0** norm, which helps us to find number of non-zero elements in the inverse vector.

$$\widehat{X} = \text{argmin } \|X'\|_0 \text{ such that } \Theta X' = Y \tag{5}$$

But unfortunately when we try to find a solution using equation 5, **ℓ_0** norm problem is NP hard

problem and numerically unstable.

ℓ_1 minimization approach: In compressed sensing ℓ_1 norm can reconstruct K-sparse signals and can predict the sparse signals with high likelihood using only $M = cK\log\frac{N}{K}$ measurements,

$$\hat{X} = \text{argmin } \|X'\|_1 \text{ such that } \Theta X' = Y \tag{6}$$

This this is known as basis pursuit and the complexity of this problem is O (N^3) [13].

2.4 Analog to information Converter:

Advances in Digital Signal Processing (DSP) has numerous applications in the field of wireless communication, multimedia, biomedical and in radar detection systems. Digital Signal Processing converts any signal to digital domain for the processing part and we do this analog to digital conversion to get rid of complex design consideration of analog handling like noise figure, feed through and linearity and that is where the analog to digital converter(ADC) comes in to picture. ADC does sampling by following Nyquist theorem, which states that ADC has to take samples at the rate of twice of the signal bandwidth to avoid the loss of any information. However there are physical limitations of ADC, in many applications like radar detection system or wireless communication which uses signals in the range of gigahertz (GHz), using an ADC for such a signal will require very high sampling rate(greater than 3GHz) and resolution of more than 16 bits that enormously surpasses current abilities. Taking into account the current scenario it could be decades before ADCs of such capacities come into existence. In many cases, interested signals having some additional structure like sparse or compressible in specific domain (Fourier or Wavelet), and sparse signals on an average has very few information contained in; most of the RF application often has very large bandwidth but very less “information rate” [7]. Sampling of

this sparse signal at Nyquist rate neglects extra information, so uniform sampling is not an exceptionally productive technique for information extraction. Over the last one decade, another hypothesis of compressed sensing (CS) has developed to empower signal sampling past Nyquist limitations for the specific class of signals.

The principle thought of CS is to reconstruct the signals from the very few measurements than the required by the conventional theorem for the specific set of signals. Using the CS hypothesis we can design an Analog to Information Converter (A2I) which can be used to sample the signal and also we can recover the signal using reconstruction algorithm subsequently reducing numerous issues of traditional ADCs [9]. In addition to that, it sends the almost similar information at the sub-Nyquist rate. The block diagram of the Analog to information converter (AIC) is shown below:

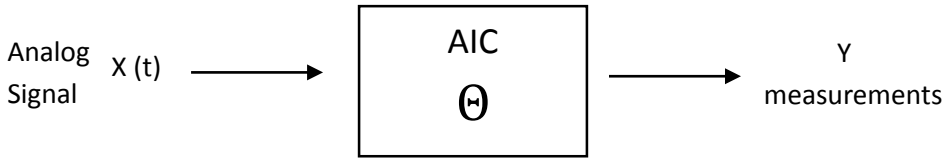


Fig. 2.3(a) Measurement matrix Θ converts analog signal to the digital measurements at Sub-Nyquist rate [7].

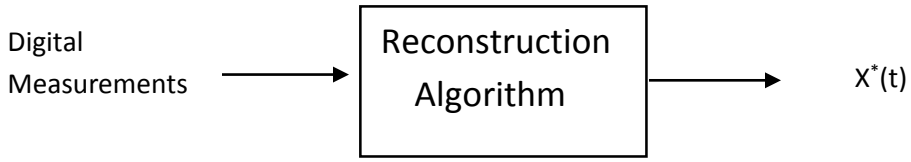


Fig. 2.3(b) Algorithm reproduces the signal which is similar to X (t) from the information stored in Y measurements [7].

Up until now, the basis of compressed sensing is discussed and the basic architecture block diagram of Compressed Sensing Analog Front end Design (CS-AFE) is shown below.

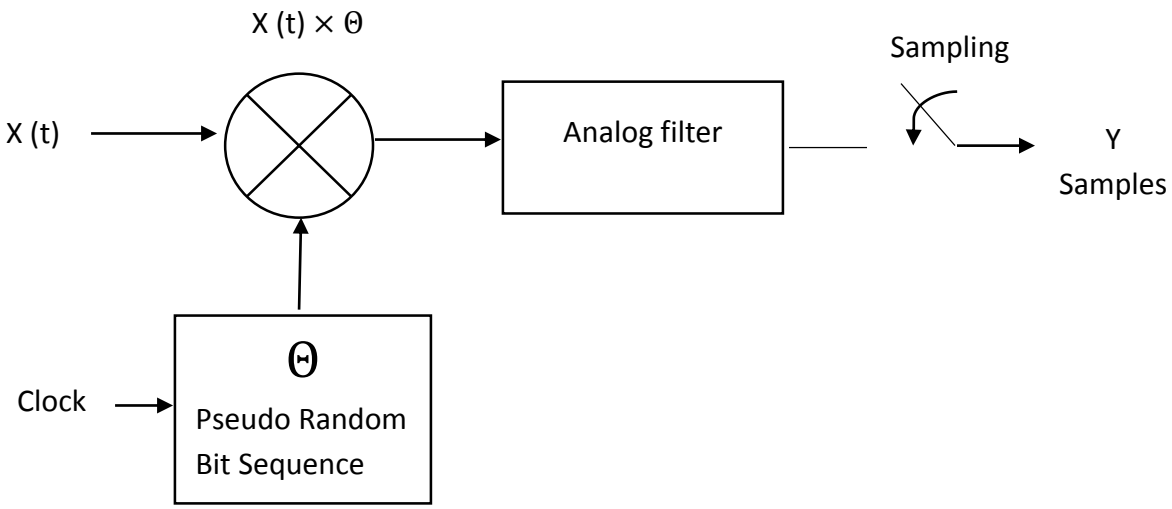


Fig. 2.4 Architecture of AIC [7].

In the next section we will discuss the previous architecture design presented for the Analog to information converter.

2.5 Random Demodulator:

In 2007, a research team from the rice university presented the first analog front end design for the compressed sensing and named it Random Demodulator (RD). RD uses Pseudo Random Bit Sequence (PBRS) to generate random sequence which can be used as a measurement matrix and after that it multiplies input signal which must be sparse or compressible with the random sequence generated by PBRS. RD uses Maximal-Length Linear Feedback Shift Register (MLFSR) as a PBRS, it generates sequence of $2^n - 1$ bits for n number of shift registers. It randomly generates

± 1 with zero average. Semi dynamic flip-flop (SDFF) is used in the shift register [9]. RD was designed in 130nm CMOS technology and their design was working properly up to 2 GHz.

The block diagram of the Random demodulator is shown below:

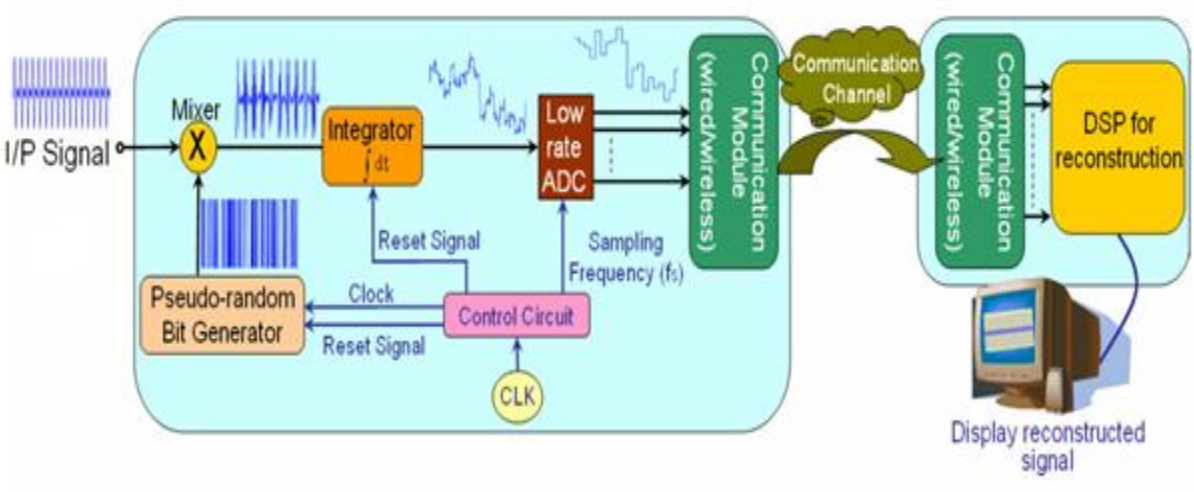


Fig. 2.5 Hardware block diagram of RD [9].

In RD for multiplication of PRBS and input signal, modified gilbert mixer is being used. The original gilbert mixer is redesigned to improve its frequency response and minimize the nonlinearity. For analog filtering an integrator is being used as a low pass filter, and differential input differential output amplifier is used for RC- Integrator [8]. It has finite time constant due to low gain of the amplifier and that can be decided from values of R and C. Integrated output from the integrator is sampled using low rate ADC and samples are directed to the back end to reconstruct it and get the original or a signal similar to the original Using RD, sampling of data is being done data at $1/8^{th}$ the rate of the Nyquist rate [8].

2.6 **RMPI:**

RMPI stands for RANDOM MODULATION PRE INTEGRATION, is an analog front end architecture designed by a research team of California institute of technology in 2009. It is designed to overcome the drawbacks of the random demodulator. Block diagram of RMPI is shown below

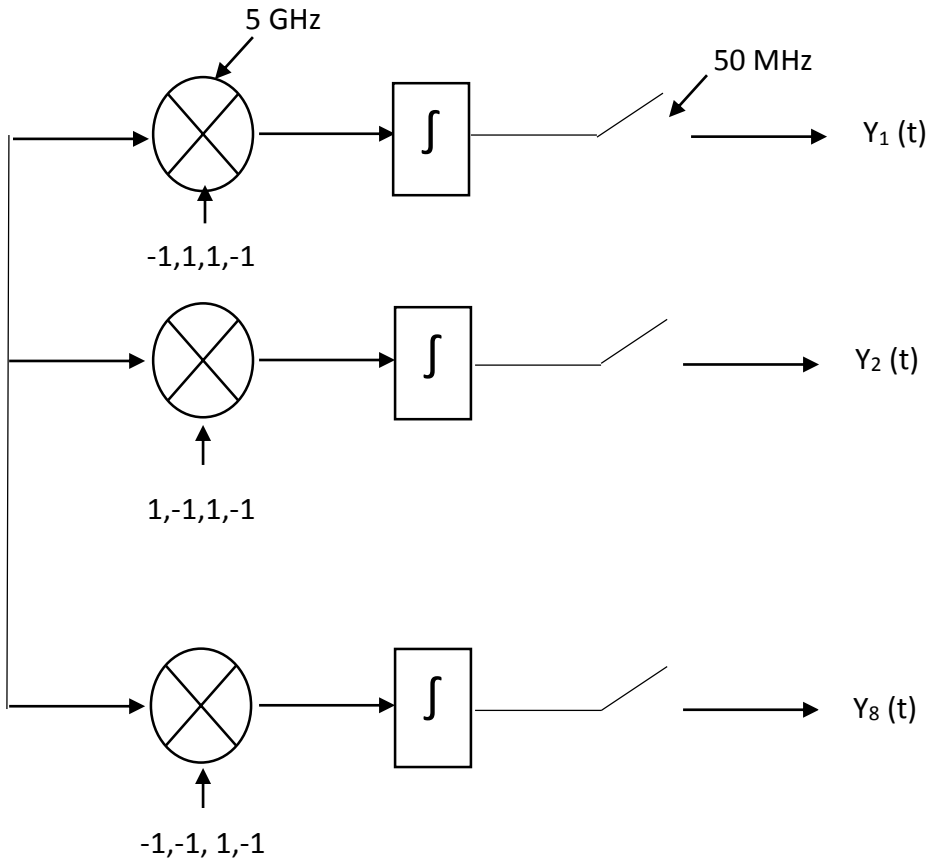


Fig. 2.6 Architecture block diagram of RMPI [3]

RMPI also uses the concept of random demodulator, it uses different parallel channels instead of modulating entire signal in one channel. First all inputs will go to Low Noise Amplifier (LNA) to amplify the input and reduce the noise figure. Output of the LNA will go to different parallel channels, here each channel gets the same input [3]. Trans-conductance amplifier is connected in the beginning of the every channel to reduce the crosstalk between different channels where output of low noise amplifier is altered in to current signal and sent to mixer for modulation. In

each channel input signal is modulated with different PBRS sequence and then sent to integrator, once integration is done integrated signal will be sent to do ADC. Sampling will be done at sub Nyquist rate using ADC [2].

In RMPI 4GHz is used as modulating frequency, so input signal can be of any frequency up to 2 GHz with dynamic range of more than 54 dB [2]. RMPI is capable of doing under sampling at 12.5 time's lower rate than Nyquist rate. [4] RMPI uses 128 bits repeated sequence as a measurement matrix which is modulated with input signal using a passive mixer. Class A Op-amp based TIA RC integrator is used for integration. Each ADC in RMPI does under sampling at the 40 MSPS and overall sampling rate is 320 MSPS (4 GHz/12.5), due to parallel channeling structure, interleaved ADC comes in to the picture because at one point of time only one ADC does sampling and all other ADCs will be off [6].

Chapter 3

Random Demodulator in TSMC 180nm CMOS Technology

In this thesis, we have proposed a new circuit design of AIC, which uses random demodulator in the TSMC 180nm CMOS technology. The circuit is designed and stimulated in Cadence Virtuoso Schematic version 6.1.5. In this design of AIC, we use Gold Code generator as a pseudo random sequence for the modulation of the input signal.. The Gold Code generator is designed to do modulation at 5 GHz.

3.1 Gold Code Generator

Gold code generator is a pseudo random sequence generator, which uses two Pseudo Random Bit Sequence (PBRS) to generate one gold code sequence. Gold Code is utilized broadly in Code Division Multiple Access (CDMA) and also in Global Positioning System (GPS). PBRS is an orthogonal, finite length binary sequence. Ideally each sequence generated by PBRS should be orthogonal to the every time shifted version of that sequence. PBRS is used to spread the input signal over the 5 GHz spectrum and each input signal can be unequally modulated using different PBRS. The main logic block of the gold code is comprised of Linear Feedback Shift Register (LFSR) and Ex-OR gate.

3.1.1 Types of Linear Feedback Shift Register

There are two types of LFSR.

1. Galois implementation
2. Fibonacci implementation

Galois implementation:

The Galois implementation is also known as m-type LFSR in which the flowing of data is done from left to right and feedback is done from the right side to left side and polynomial order of shift register increments from left to right starting from X^0 [11]. Figure 3.1 displays block diagram of the galois implementation.

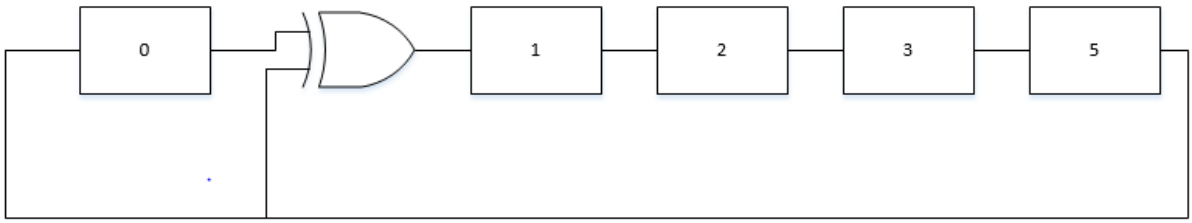


Fig. 3.1 LFSR using Galois Implementation [11]

Here in Galois Implementation output of one shift register is going as an input of next shift register at every positive edge of the clock and at the taps, where output of shift register is going as an input to the Ex-Or gate with the output bit before going in to the input of next shift register.

PBRS equation generated by galois implementation is

$$F(X) = X^5 + X^1 + 1 \quad (1)$$

Fibonacci implementation:

The Fibonacci implementation is most commonly used type of LFSR, It is also known as Simple

type or out of line LFSR. In this implementation data flowing is done from left to right and feedback path is reverse of that, right to left. The bit positions that have an impact on next input bit are called taps [11]. The furthest right bit is called output bit and polynomial order of the shift register increments from right to left. Figure 3.2 displays the block diagram of the Fibonacci implementation.

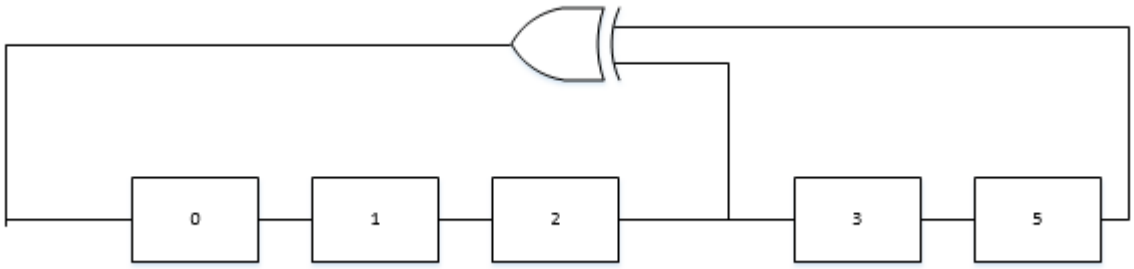


Figure 3.2 LFSR using Fibonacci Implementation [11]

In Fibonacci implementation output of each shift register is an input to the next shift register and new bit is generated by Ex-ORing of all taps with output bit and it goes to the input to the left most shift register. Total number of bits generated by LFSR before repeating the same sequence are $2^l - 1$, where l is number of shift registers. The equation of the sequence generated by the figure shown above is

$$F(X) = X^5 + X^2 + 1 \quad (2)$$

The objective of communication system can be achieved by making the LFSR to follow the following special correlation properties:

1. Auto Correlation
2. Cross Correlation

Auto Correlation:

Auto correlation, also known as a serial correlation is mathematical representation of how well time Signal f (t) can differentiate between itself and a lagged version of itself [11]. For a finite and discrete signal auto correlation can be define as a following equation:

$$r_{xx} = \sum_{n=0}^{L} x(n)x(n-T) \quad (3)$$

Here L is a length of a sequence and T is a time delay. From the equation we can say that if auto correlation is positive then it is hard to distinguish the original sequence from the time shifted version. If auto correlation is negative then it means that signal can be easily distinguished from its time shifted version and if auto correlation is zero it states that time shifted version is orthogonal to the original signal [11].

Cross Correlation:

Cross Correlation is a measure of at what extent two signals are correlated to each other [11]. Suppose two signals are x (t) and y (t), then Cross Correlation is defined by following equation

$$r_{xy} = \sum_{n=0}^{L} x(n)\, y(n-T) \quad (4)$$

In the following table, cross correlation between two m- sequence and two gold code sequence are given for different l values. The results shows that Cross Correlation between two gold code sequence are less than regular m-sequences, to reduce the co-channel interference between two channels we want Cross Correlation to be as small as possible.

Table 3.1 Cross Correlation between m-sequence and Gold Code

Number of Flip flops in LFSR	$m = 2^l - 1$ Number of bits in m-sequence	Cross correlation Of m-sequences	Cross correlation Of Gold code
3	7	0.71	0.71
4	15	0.60	0.60
5	31	0.35	0.29
6	63	0.36	0.27
7	127	0.32	0.13
8	255	0.37	0.13
10	1023	0.37	0.06
12	4095	0.34	0.03

3.1.2 Implementation of Gold Code in Cadence Virtuoso:

Two main components in the gold code are Ex-OR gate and Shift Register. Firstly in this section new architecture of the Ex-OR gate is presented which has very less delay compared to the conventional Ex-OR gate and in next section we will discuss the architecture of the shift register. First we will start with differential switch that we need later to build Ex-OR gate. The architecture of differential switch is discussed below.

3.1.2.1 Differential Switch:

The schematic diagram of the differential switch is shown below in the figure 3.3. It consists 4 NMOS transistors, the main idea behind this design is to avoid use of PMOS transistors because of longer switching time. In the schematic of the differential switch, there are 2 inputs which are In and In_b, Vc and Vc_b are the control line and Op and Op_b are the output of the differential switch. The Symbol of the differential switch is shown below in the figure 3.4, direction of the arrow shows the current direction. In differential switch when you give (Vc, Vc_b) as (0, 1), it will make M1 and M4 transistor turn on and In will be connected to Op and In_b will be connected to Op_b. If you give (Vc, Vc_b) as (1, 0), M2 and M3 transistor will be turned on and In will be connected to Op_b and In_b will be connected to Op.

Table 3.2 Operation of Differential Switch

Control line	Op	Op_b
(Vc, Vc_b) = (0, 1)	In	In_b
(Vc, Vc_b) = (1, 0)	In_b	In

In figure 3.5 waveform of the differential switch at 1 GHz is shown and average propagation delay at 1 GHz is 8.5639 ps. Figure 3.6 displays waveform of the differential switch at 5 GHz and delay for that is 7.984 ps.

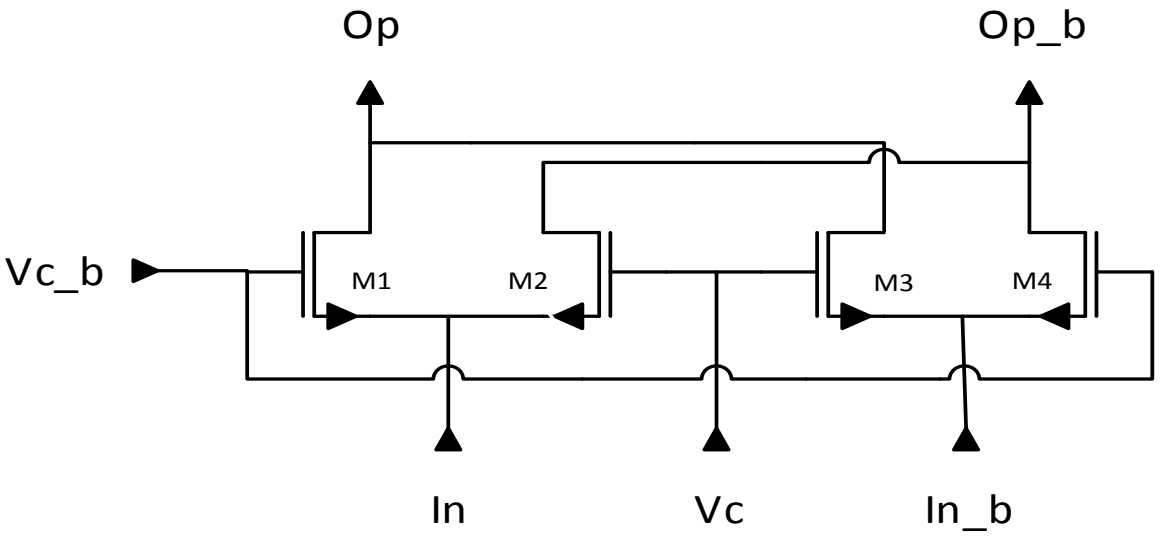


Fig. 3.3 Schematic of differential switch

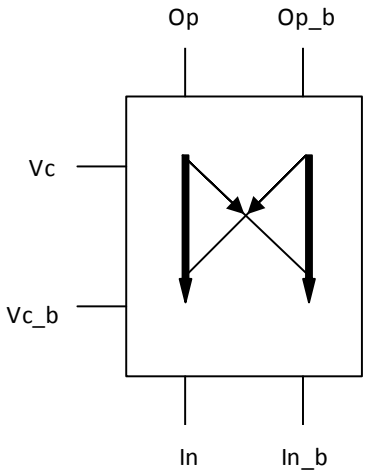


Fig. 3.4 Symbol of differential switch

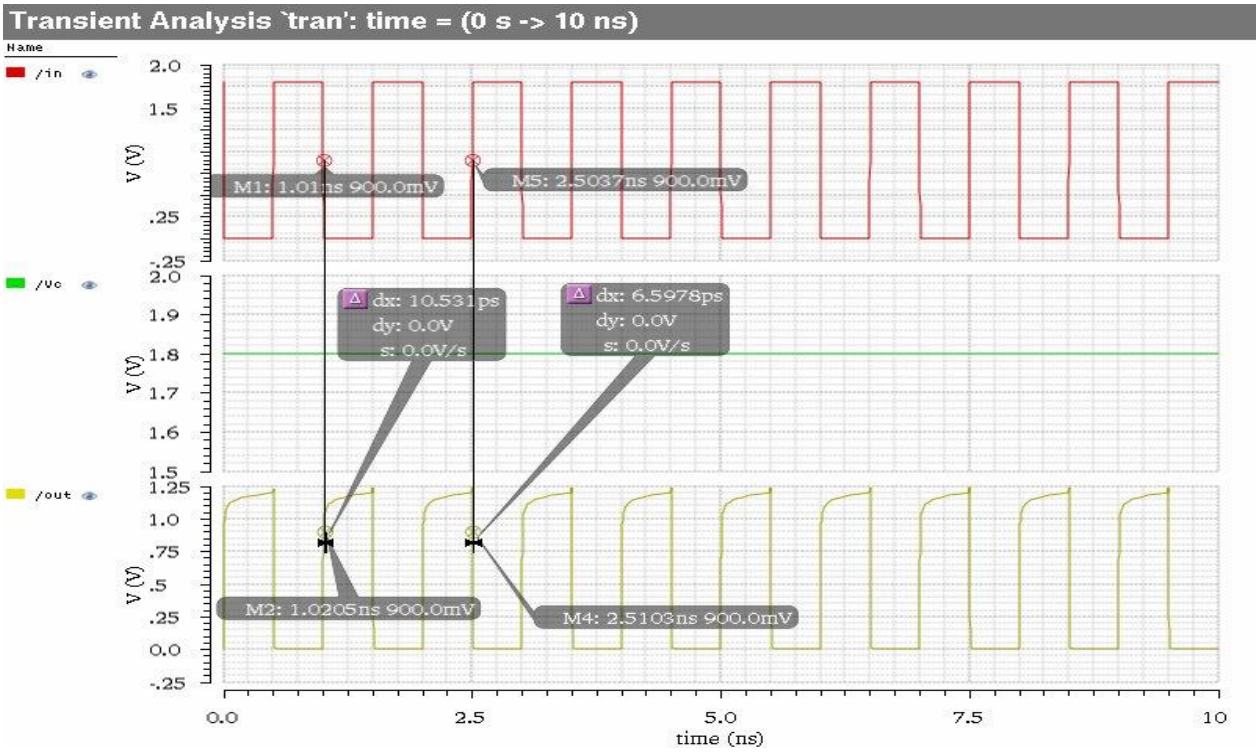


Fig. 3.5 waveform of differential switch at 1 Ghz

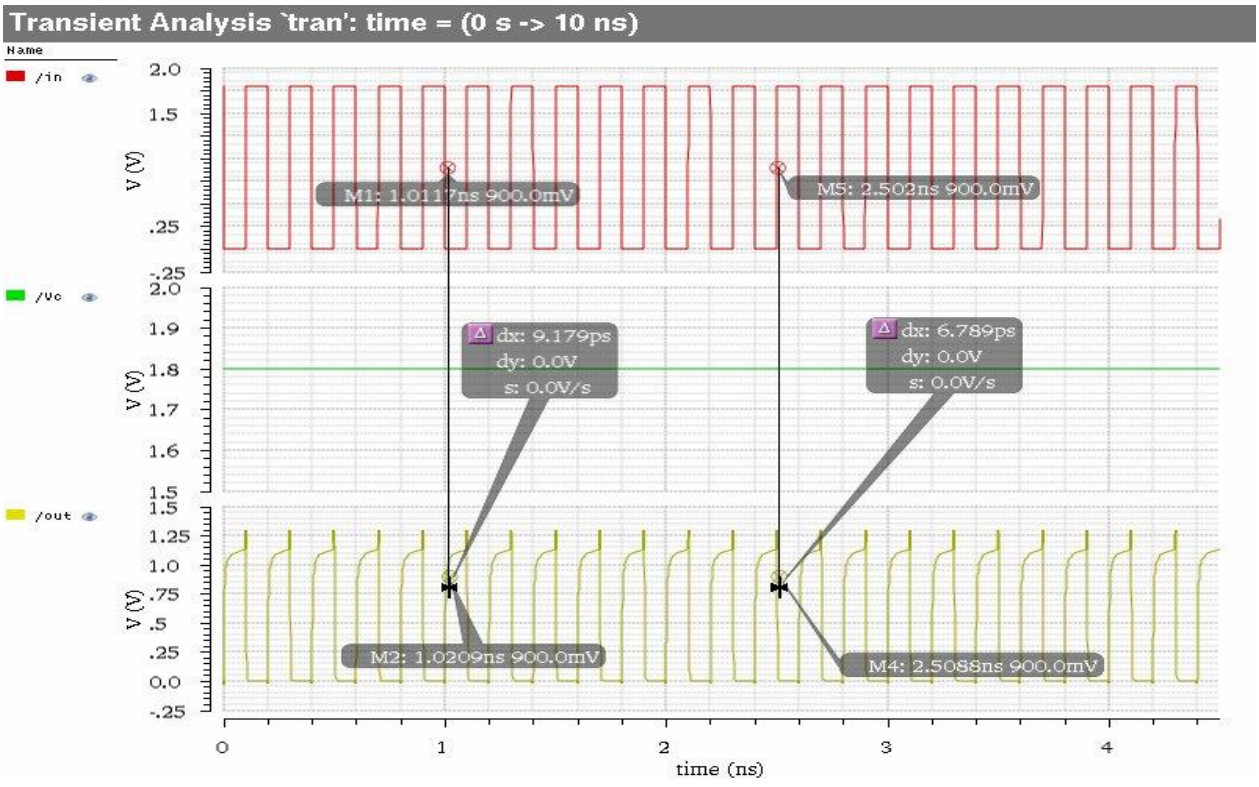


Fig. 3.6 waveform of differential switch at 5 Ghz

3.1.2.2 Ex-OR gate using Differential Switch:

In order to build an Ex-or gate we will use two differential switch and two resistors. Figure 3.6 shows the architecure of the 2 input Ex-OR gate, where (A, A_b) and (B, B_b) are inputs of the Ex-OR gate and (C,C_b) are the differential output of the Ex-OR gate. A and B are given as a control line input and voltage supplies are given as an input, when both inputs of the Ex-OR gate are same, Vdd! will get connected C_b and C will get connected to the Vss!. If both inputs are different then C will get high logic and C_b will be low logic.

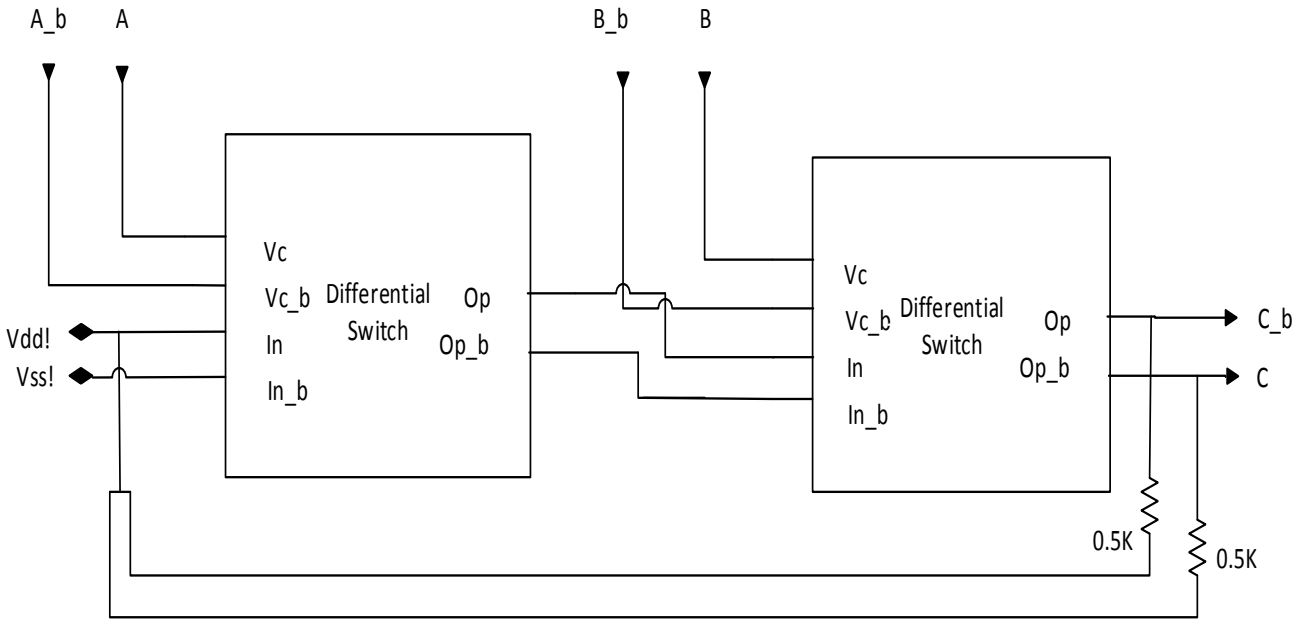


Fig. 3.7 2-input Ex-OR gate using Differential Switch

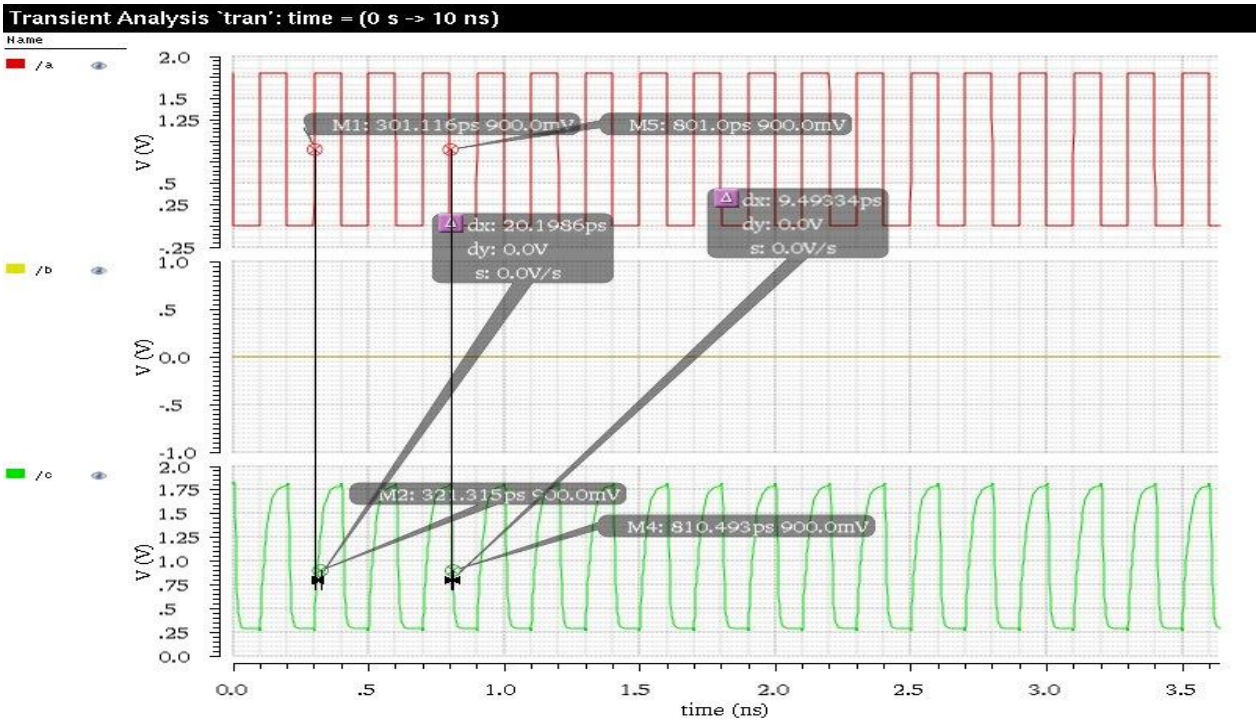


Fig. 3.8 waveform of 2-input Ex-OR gate at 5 Ghz

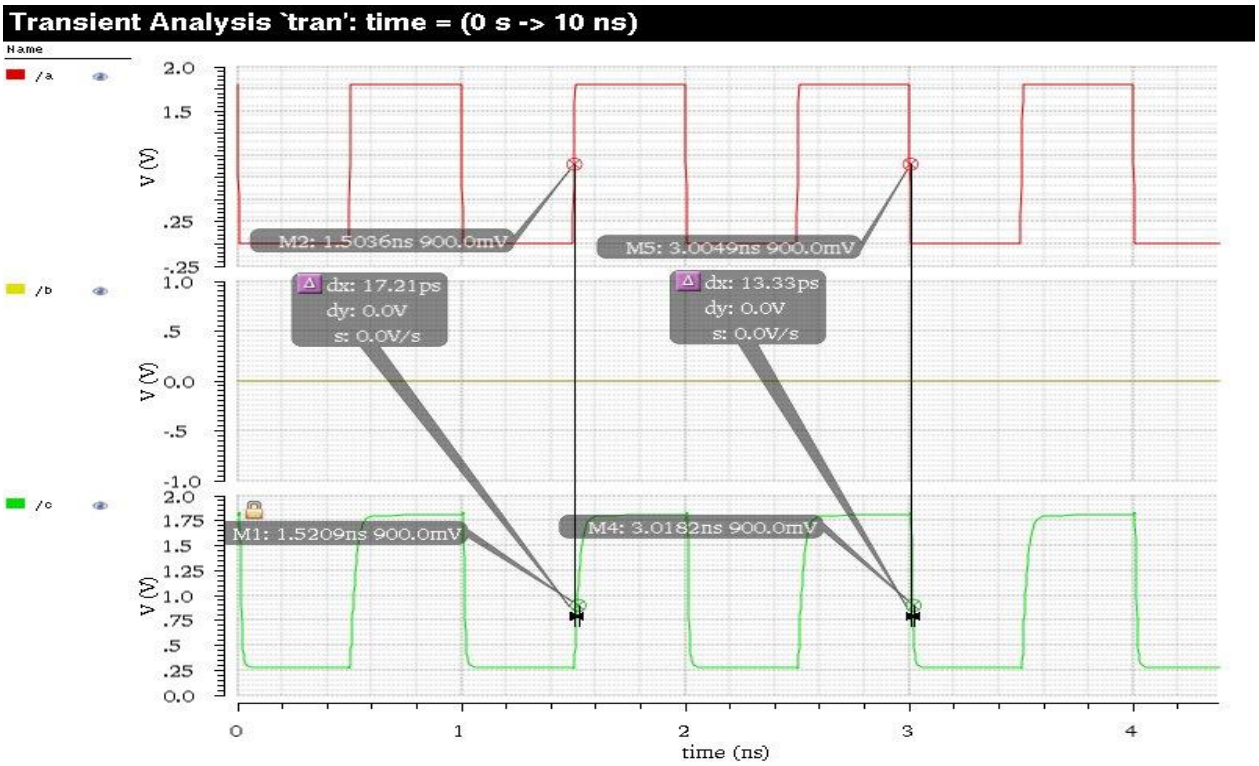


Fig. 3.9 waveform of 2-input Ex-OR gate at 1 Ghz

Table 3.3 Operation of Ex-OR

A	B	Differential Switch1	Differential Switch1	C	C_b
0	0	OFF	OFF	0	1
0	1	OFF	ON	1	0
1	0	ON	OFF	1	0
1	1	ON	ON	0	1

Figures 3.8 and 3.9 present the output waveforms of the Ex-OR gate for 1 GHz and 5 GHz as well. You can see that the performance in terms of delay of the Ex-OR is far better than conventional Ex-OR gate using CMOS. Average propagation delay of the Ex-OR gate at 1 GHz is 14.843 ps, and for 5 GHz is 15.27 ps. Table 3.3 shows the truth table of the Ex-OR gate to verify the waveforms. This design approach can be applied to multiple inputs Ex-OR gates.

3.1.2.3 CML latch:

To build shift register In the Gold Code generator we use special kind of conventional CML latch. Every shift register contains two CML latch; the first one operates on the positive edge of the clock and the second one operates on the negative edge of the clock. This shift register operates efficiently at high frequency. We use only NMOS transistors to make CML latch operate fast.

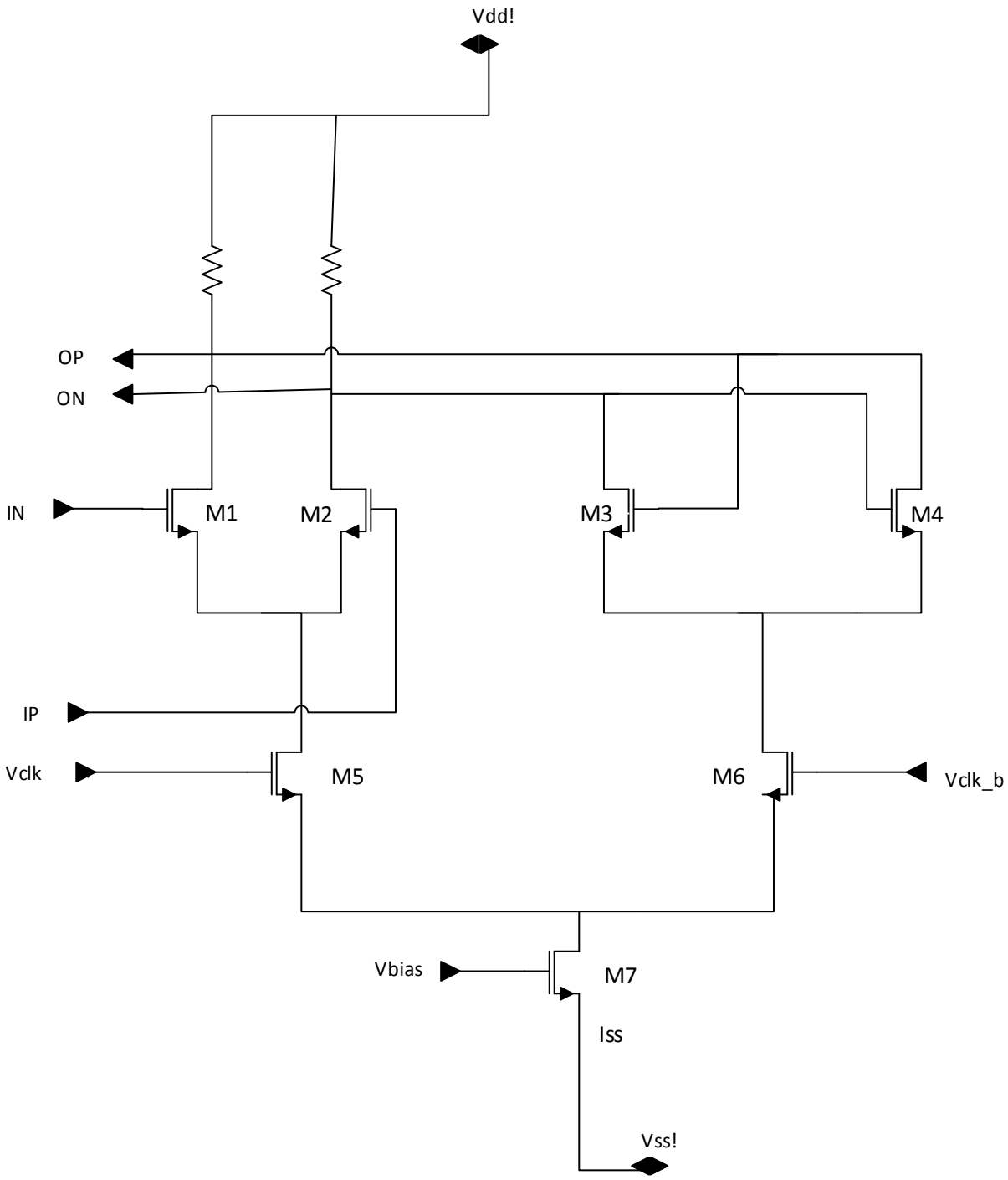


Fig. 3.10 Schematic design of CML latch

Schematic design of the CML latch is shown in the above figure, two main operations for CML latch are tracking and storing data. The tracking operation uses NMOS transistor M1 and M2. They are used to sense and track the information variation from the input and transistor M3 and M4 are used to store the data. Operation of the CML latch depends on the Vclk and Vclk_b. If Vclk is high then CML latch operates in the tracking mode; otherwise, it operates in the storing mode. When Vclk is high, it allows Iss current to flow to the tracking circuit and allows ON to track the IN. When Vclk is low then the other differential pair enables storing the logic state and output. When we first implement the above CML latch in TSMC 180 nm technology, it operates in clock frequency below 3 GHz. There are several speed limitations of this design of CML latch. A main limitation of this latch is, both tracking pair and storing pair of CML latch use the same tail current, and biasing operations of tracking and storing pairs are highly related therefore the tail current must be large enough to maintain the linearity of tracking and storing pairs. A new modified design of the CML is presented in figure 3.11. You can observe that the tracking differential pair and the storing differential pair uses two different transistors for tail current so the issue of previous design as mentioned is resolved. When latch is switching from tracking mode to storing mode, tail current must recharge the capacitance of the M1 and M2 otherwise it will draw current from the output node ON and OP. To avoid this problem the transistor M7 is connected with M5, the tail current is then passed through both transistors and when it changes from one state to another state, it helps in reducing the current spikes. Both transistors M3 and M4 are always turned on therefore we won't have any current spikes. However, the new design consumes more

power than the previous design but it can operate at higher frequency, i.e., 5 GHz to generate the Gold code sequence.

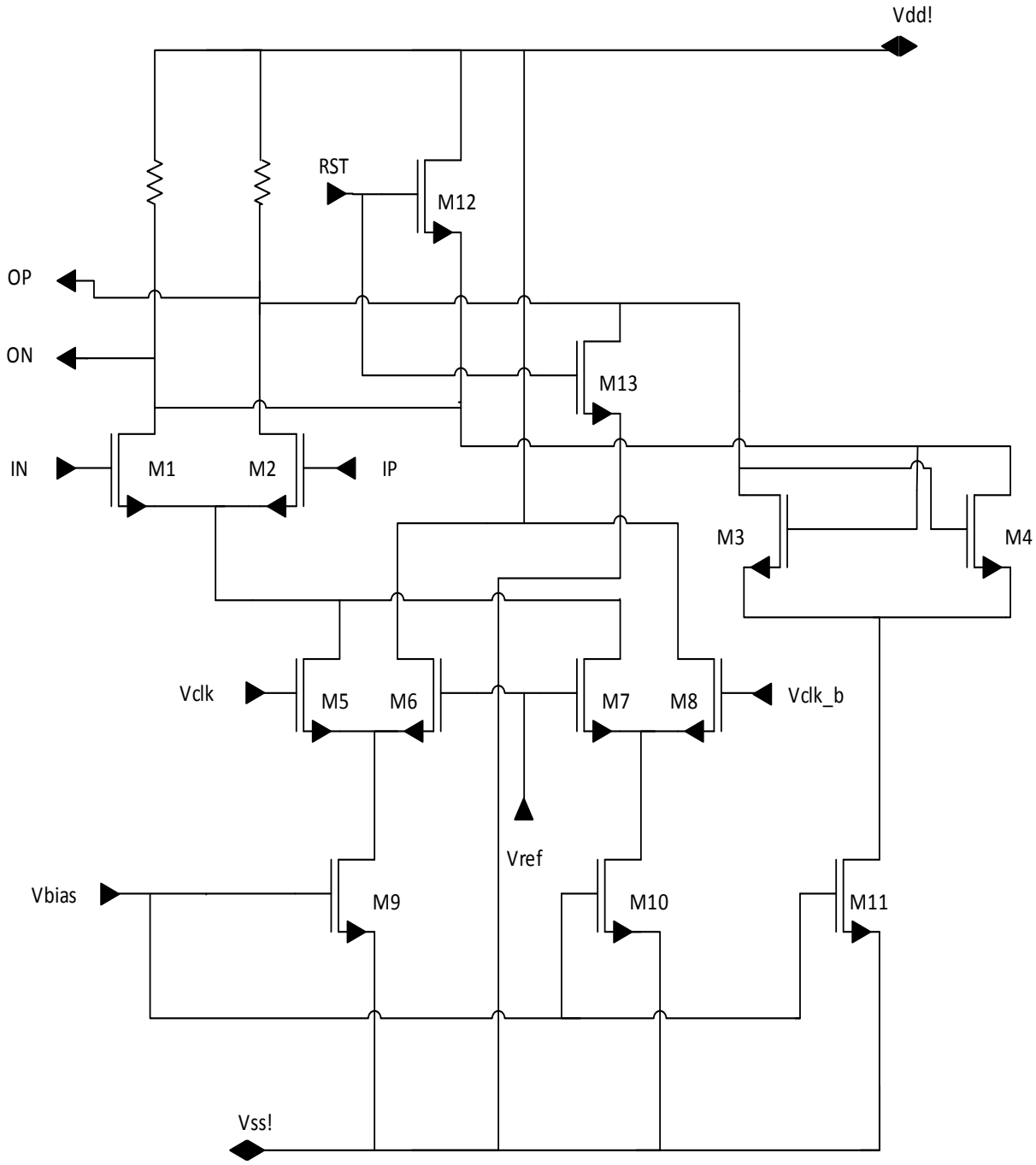


Fig. 3.11 Modified schematic design of CML latch

Other modification to the latch design is addition of a reset function, as shown in figure 3.11. When reset is high, M12 connects ON to the Vdd! and M13 connects OP to the Vss!. So it makes ON “High” and OP “Low”. We add reset function in latch when we use it in the Gold code generator. We have designed D flip-flop using two CML latchs connected together. The clock of the first CML latch is connected to the clock_b of the second CML latch and the clock_b of the first CML latch is connected to the clock of the second CML latch. Figure 3.12 presents the block diagram of the D flip-flop.

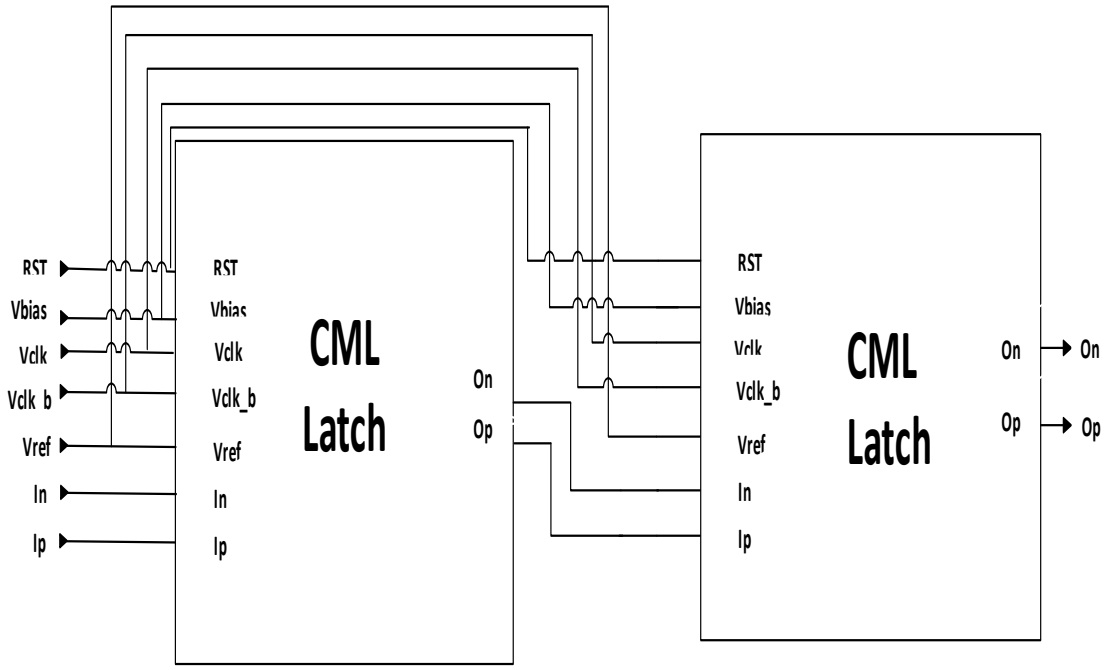
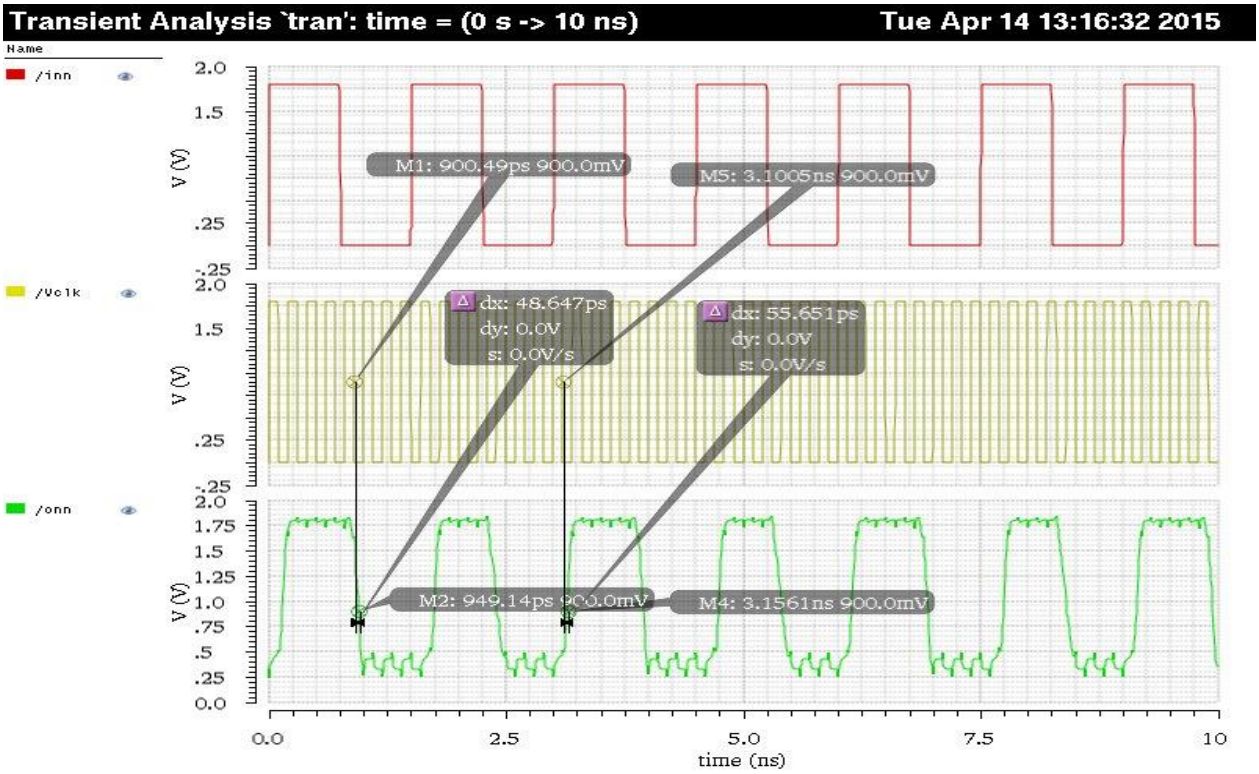
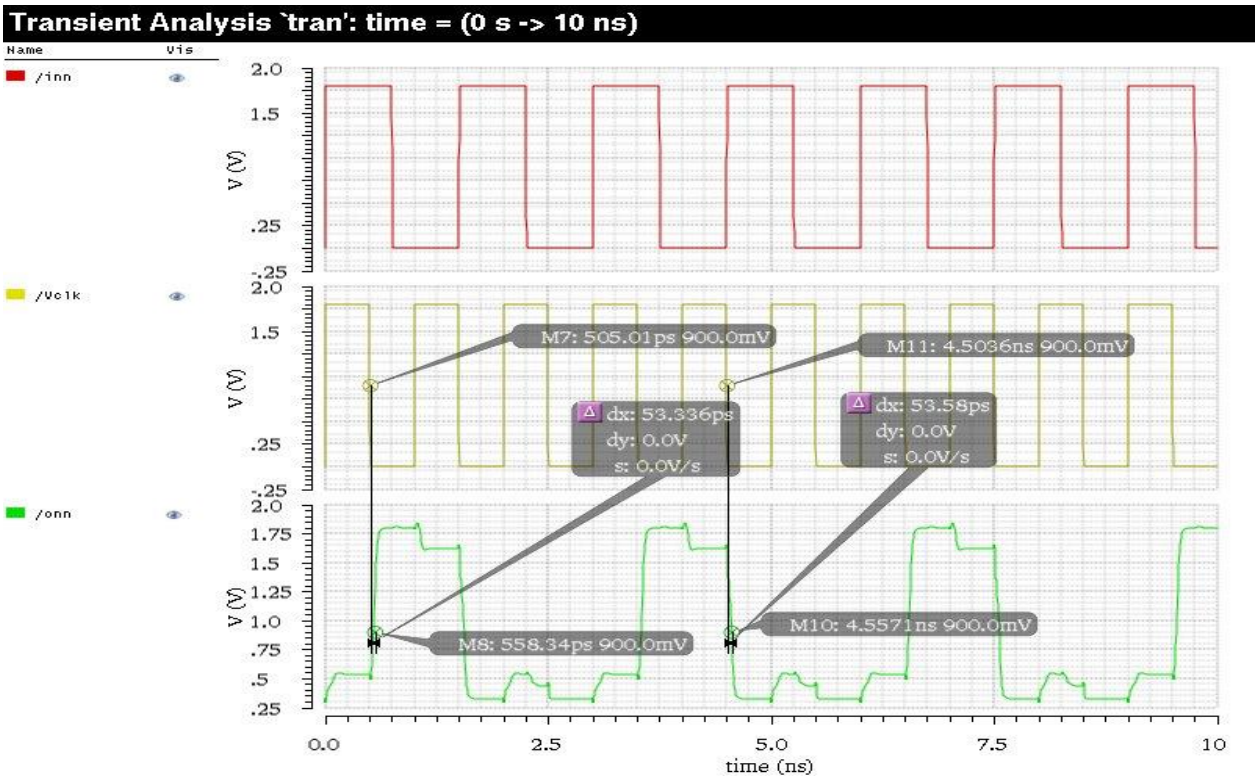


Fig. 3.12 D flip-flop using two CML latch



3.13 Output of D flip-flop at 5 GHz



3.14 Output of D flip-flop at 1 GHz

The waveforms of D flip-flop are presented. Fig. 3.12 shows waveform at 5 GHz and 3.13 shows waveform at 1 GHz. As shown in the waveform, the average propagation delay of D flip-flop is 53 ps at 5 GHz. As discussed earlier, two main components of gold code generator are D flip-flop and Ex-OR. Both D flip-flop and Ex-OR gate operates at 5 GHz. The gold code generator is then designed to operate at 5 GHz to modulate input signal at 5 GHz. Fig 3.14 shows the schematic design of the Gold code generator, which use two LFSR, LFSR-1 and LFSR-2. Both LFSR's contain 5 D flip-flop. LFSR-1 is denoted as (5, 3) and LFSR-2 is denoted as (5, 4).

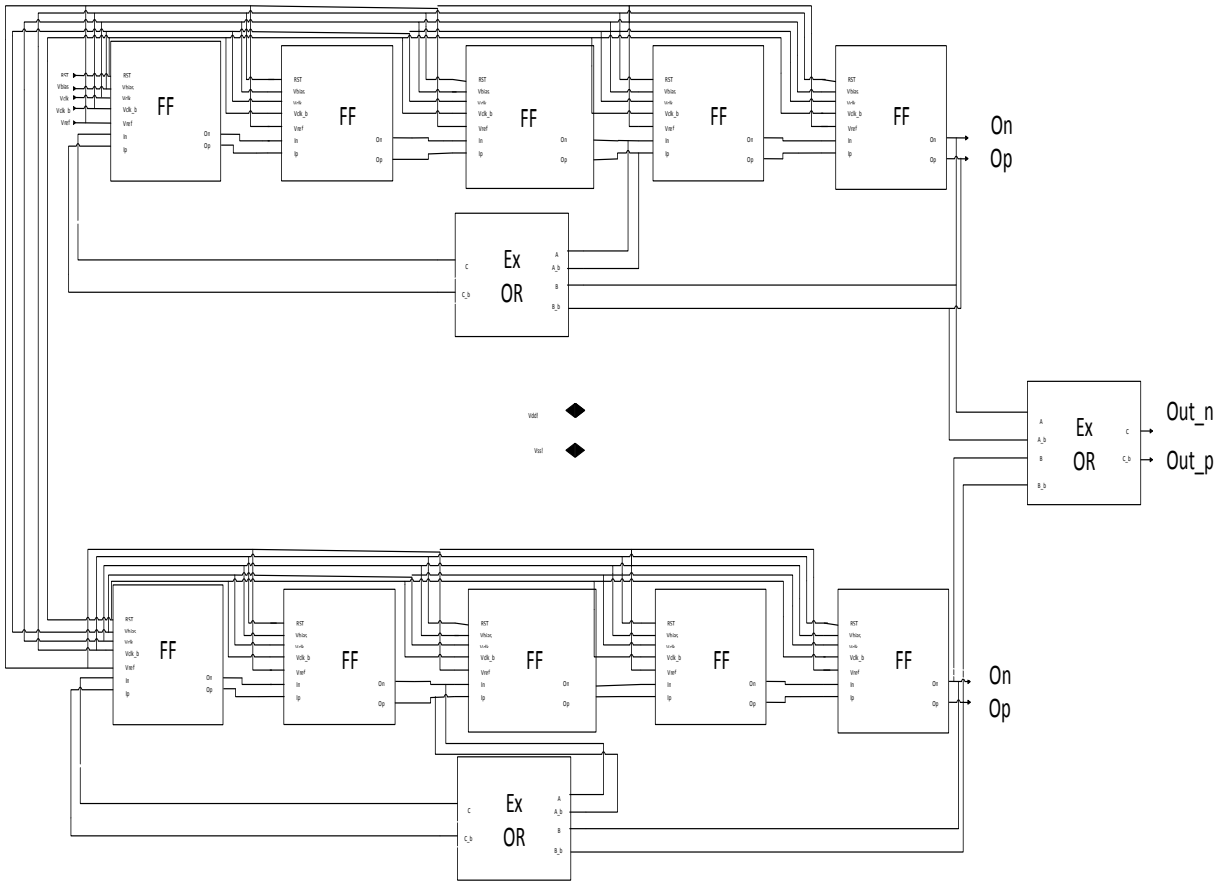


Fig. 3.14 Gold code architecture

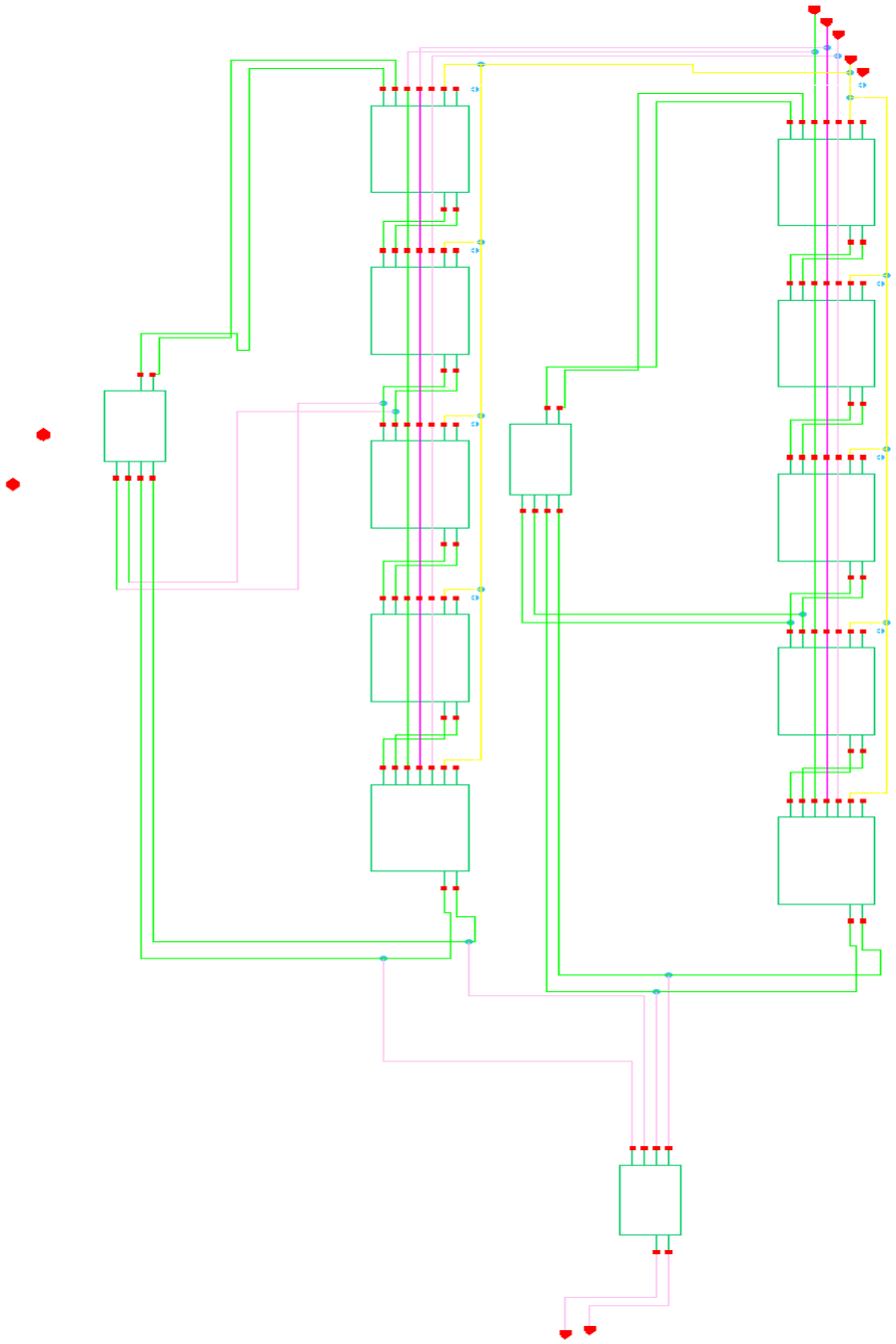
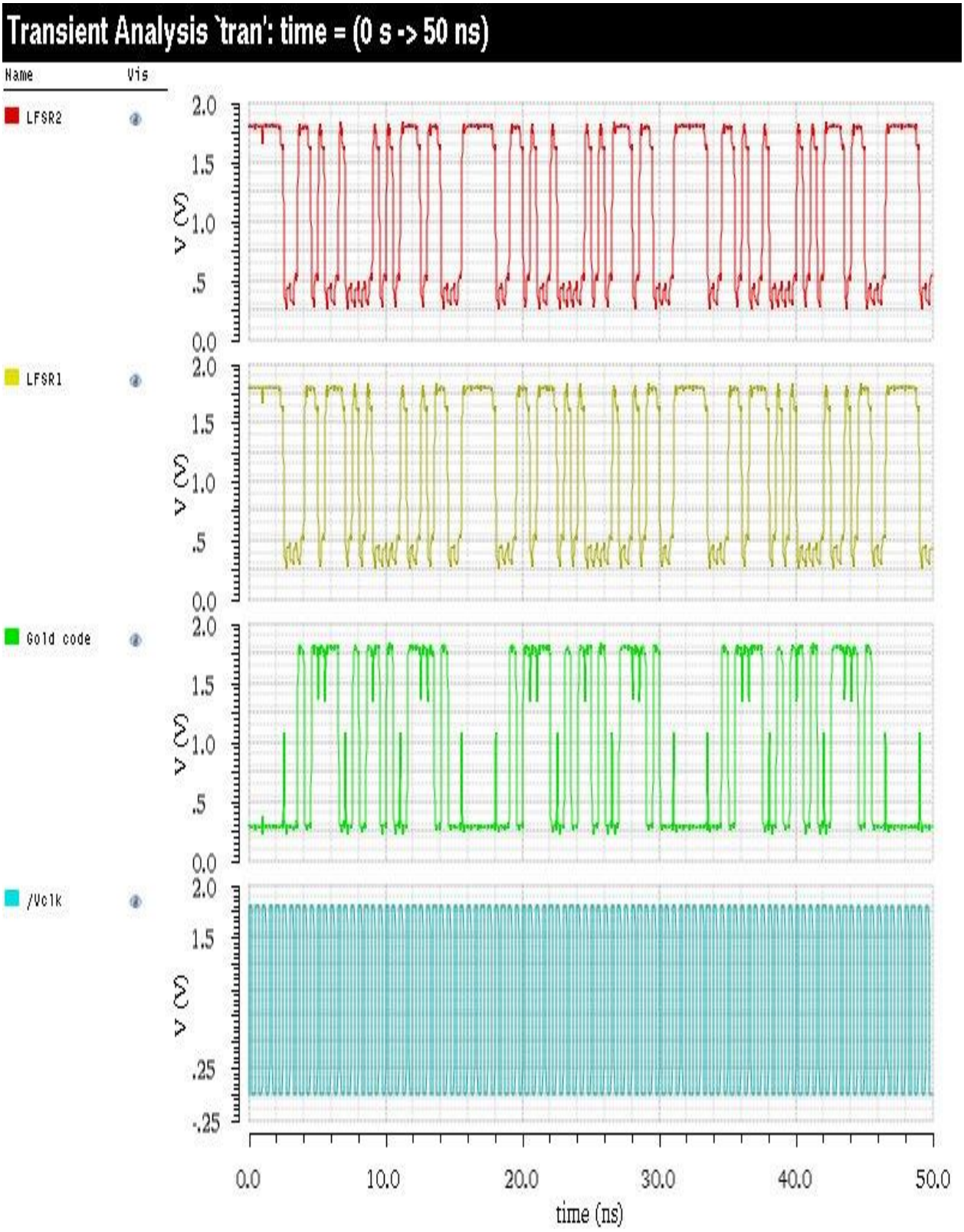


Fig 3.16 Schematic design of Gold Code in Cadence Virtuoso



3.17 Gold code output at 5 GHz

Figure 3.15 represents the schematic of the Gold code generator built using CML latch and 2 input Ex-OR gate. The block diagram is shown in figure 3.14. The output waveform generated at 5 GHz using the Gold code generator is shown in figure 3.16. It is generated using cadence Virtuoso in TSMC 180 nm technology. To verify the Gold code function of the cadence design we develope a VHDL Gold code generator and test bench. We run VHDL code of Gold code generator in Xilinx and compared the output of it with the cadence design output. Both Gold codes generate same patterns. The VHDL code and test bench are given in Appendix.

3.2 1-bit Multiplier

The 1-bit multiplier multiplies input signal with sequence generated by the Gold code generator. The Gold code generator generates a bit sequence, which contains ones and zeros. We consider bit '0' as '-1' and bit '1' as '+1'. So, we modulate the input signal with the bit sequence. And, the multiplier operates at the same speed as the Gold code generator, i.e., 5 GHz. We multiply input signal with 5 GHz bit sequence so we are able to take input signal of frequency up to 2.5 GHz. The multiplier will pass the original signal to the output if the Gold code sequence bit is '1' and it will invert or reverse the polarity of the input signal to the input if the sequence bit is '0'. Diodes and FET mixers are passive mixers. They have good linearity and less noise but consume high power and generate low voltage gain. The mixer (1-b multiplier) which we have used is presented in figure 3.17. Only NMOS transistors have been used to make the mixer circuit operate efficiently at high frequency.

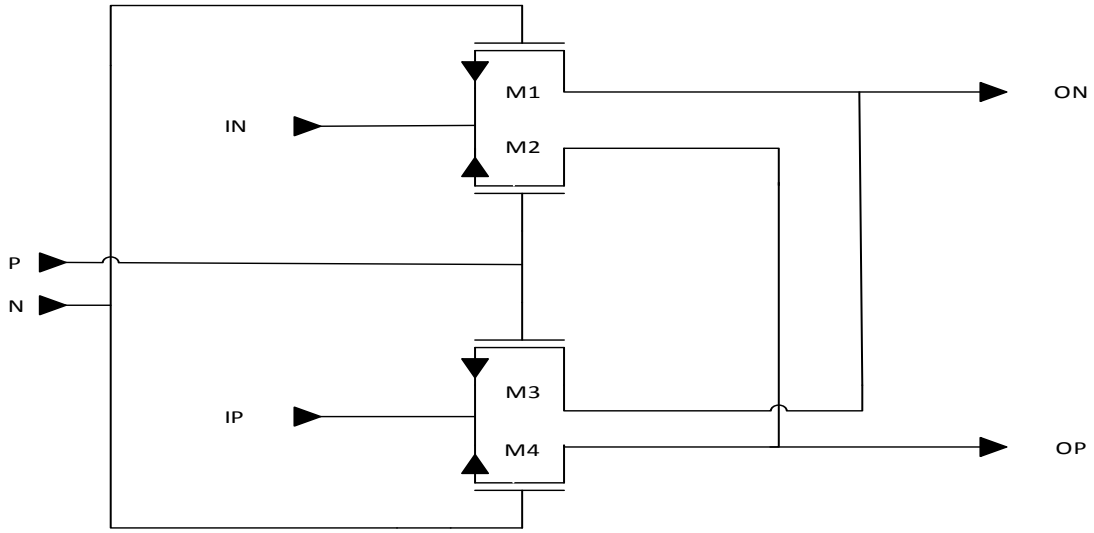


Fig 3.18 Schematic design of 1-bit Multiplier

As you can see from the design, multiplier needs differential input for both gold code generator and main input signal as well. In the figure 3.17 P and N are the input where we connect output of the Gold code generator. IN and IP are the main Input signal and ON and OP are the differential output of the multiplier. Table 3.4 shows the mixer operations. When P is high, IN is connected to OP and IP is connected to ON. When P is low, IN is connected to ON and IP is connected to OP.

Table 3.4 Mixer Operation

Gold code sequence	ON	OP
(P,N) = (1,0)	IP	IN
(P,N) = (0,1)	IN	IP

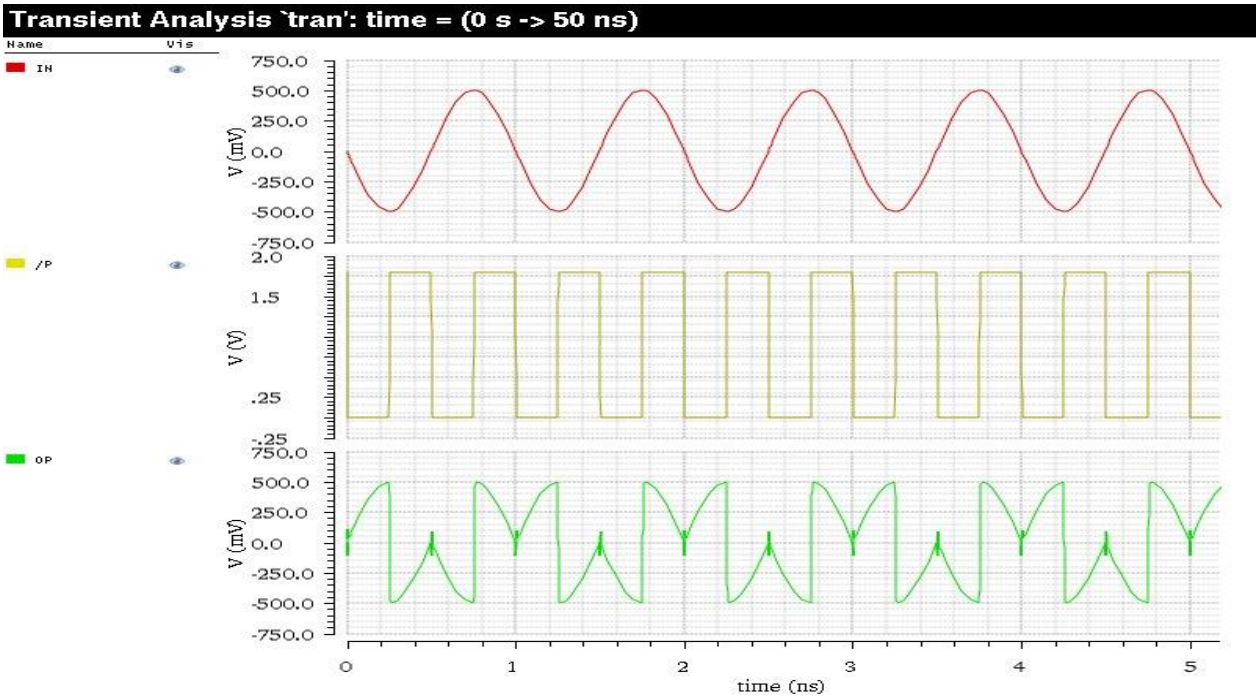


Fig 3.19 Output of 1-bit Multiplier at 2.5 GHz

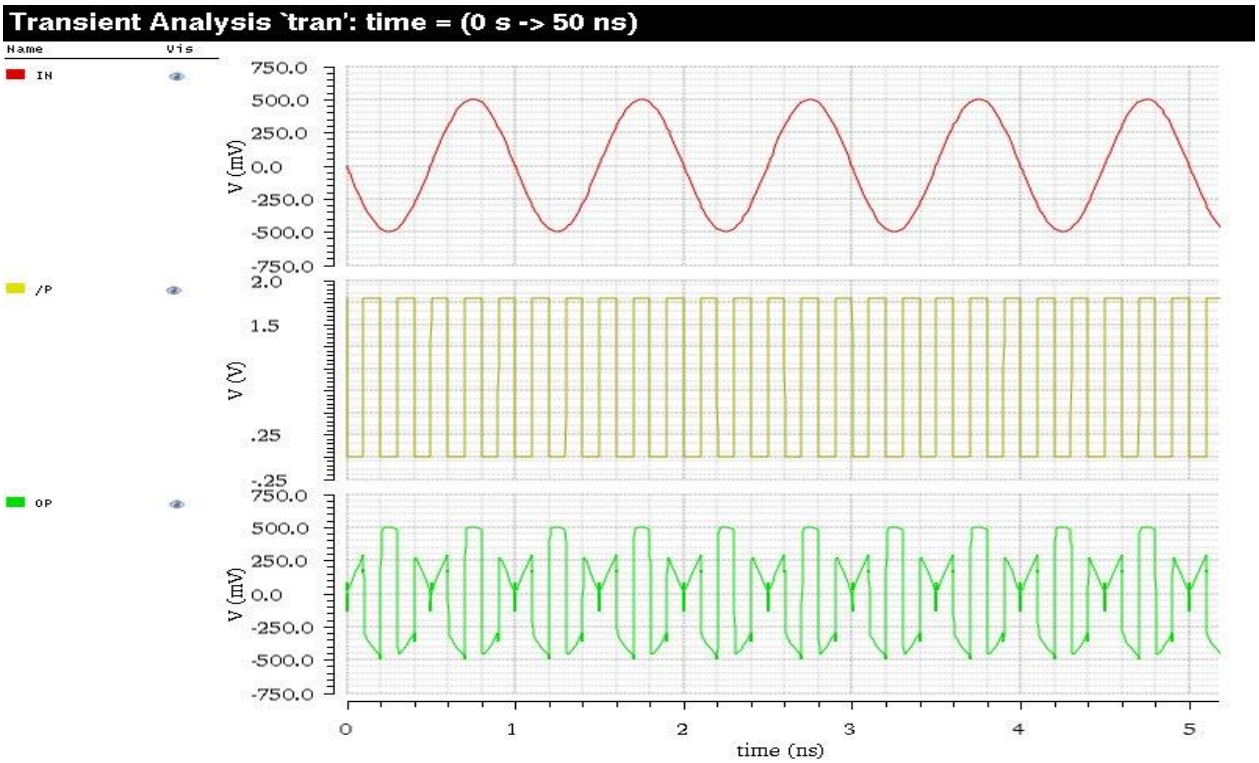


Fig 3.20 Output of 1-bit Multiplier at 5 GHz

Figure 3.19 and 3.20 shows the output waveform of multiplier in at 2.5 GHz and 5 GHz respectively.

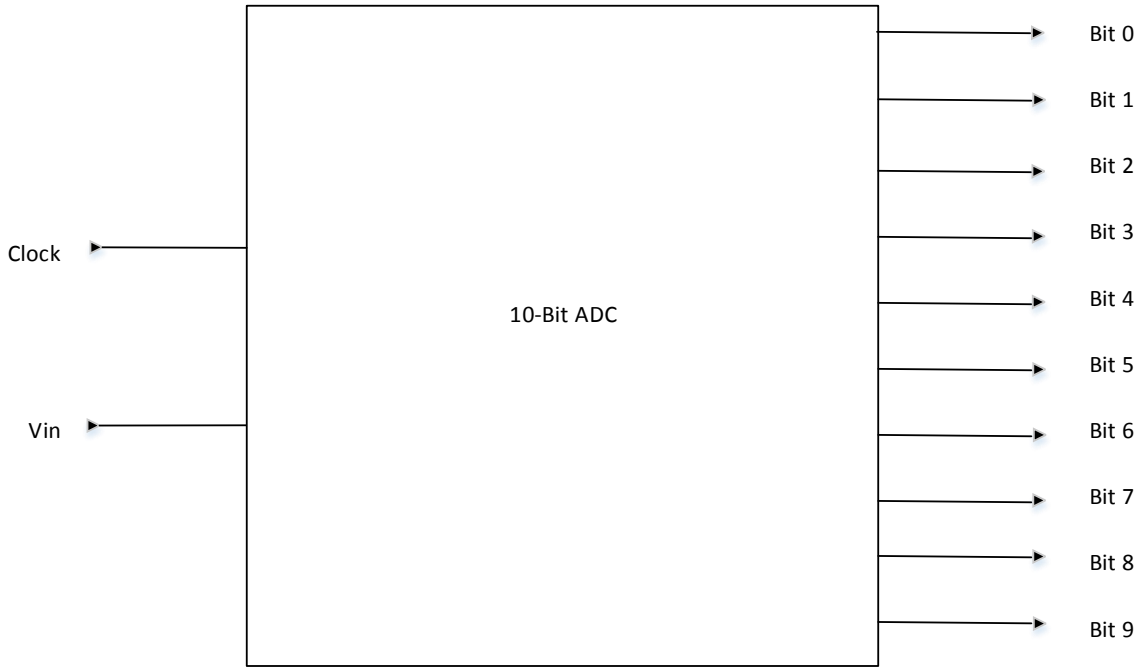
3.3 Integrator

Once we have modulated output from the mixer, it will go to the low pass filter to avoid the high frequency tones. We use Integrator which will work as a low pass filter. In this thesis we have built two different design of an Integrator. One is in Cadence, in which we used system verilog (Verilog-A) to design an integrator. I have attached the code of the integrator in the Appendix. Second design we have used for integrator is in Simulink (Matlab). We use discrete time integrator from the simulink block as an integrator, where we select reset option as an external so that we can do reset according to the system requirement. In compressed sensing we have to design Integrator such that we can reset it after every specific time. Output of the integrator will go as an input to the ADC, more explanation about the integrator is given in chapter 4.

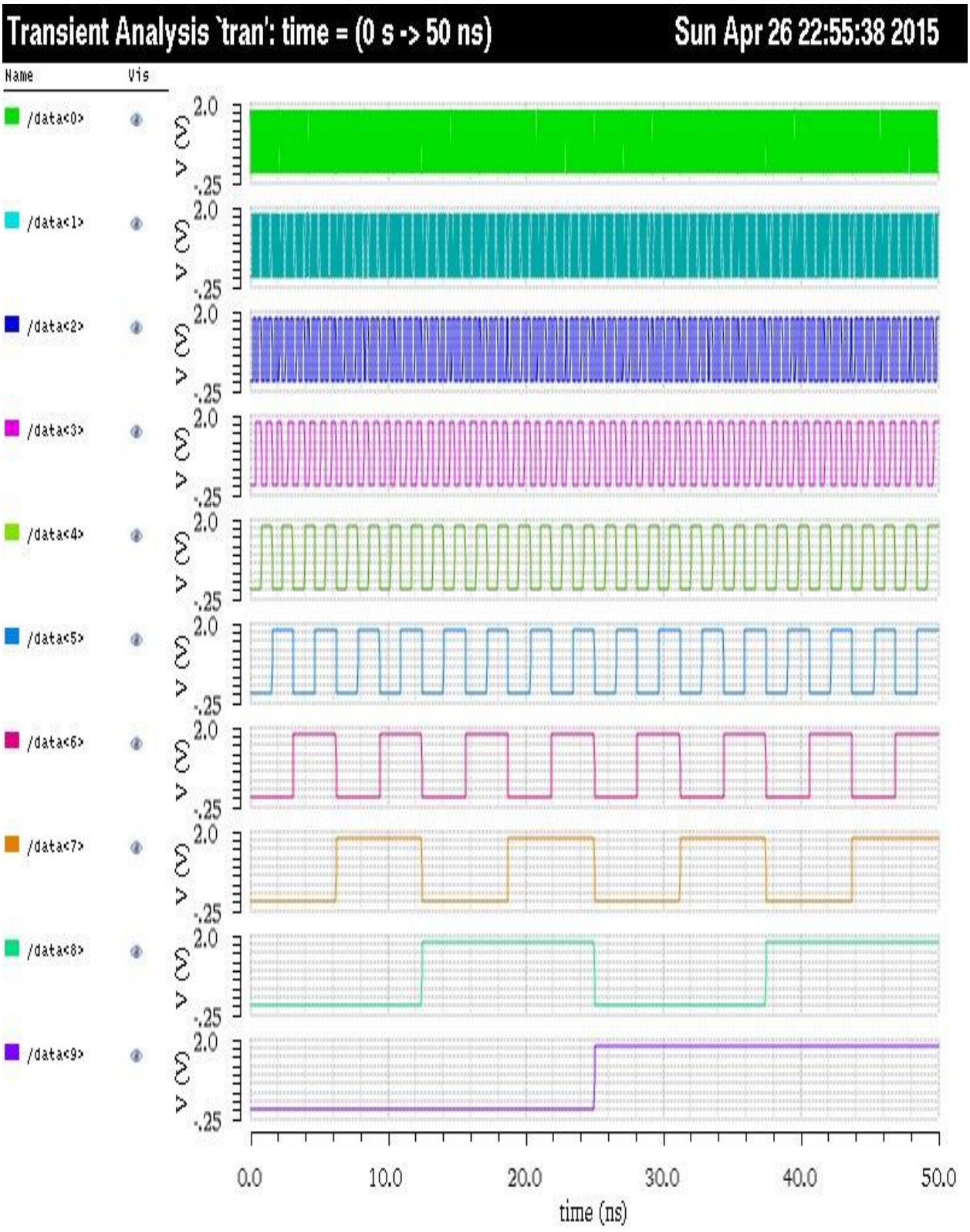
3.4 Low Sampling ADC

The output of integrator is fed to the low sampling ADC where the integrated signal is sampled at a low sampling frequency as compared to the Nyquist frequency. Our aim for this ADC is to verify the compressive sensing front-end design. We create a software model of 10-bit ADC in system Verilog (Verilog-A), which is included in Appendix. We can change parameters of ADC by just changing the variables in the code. Figure 3.21 shows the symbol of the ADC. Figure 3.22 shows waveform of the ADC, in which the input signal dynamic range is from -0.25V to 0.25V and the 10-bit ADC output sample value is from 0 to 1023. We have also designed one ADC in

Matlab, code is available in Appendix. The main idea of compressed sensing is to make ADC work at very low rare compared to Nyquist rate.



3.21 Symbol of ADC



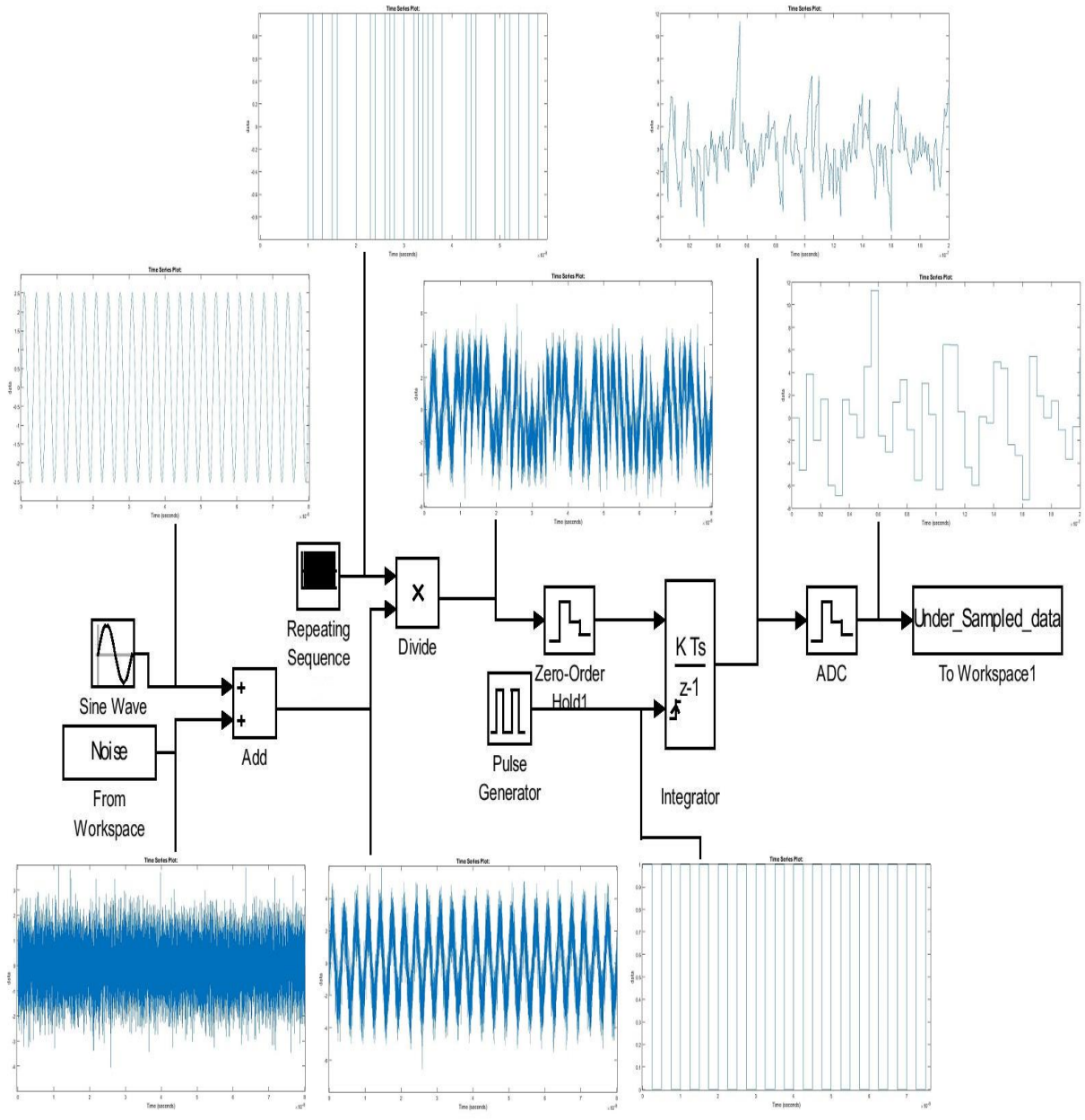
3.22 Output of 10-bit ADC

Chapter 4

Compressive Sensing Front-End (CSFE) Performance Evaluation

4.1 Ideal CSFE in Matlab:

Many have presented random demodulator designs but most of them perform at frequency less than 2 GHz. Our aim for this research is to design a random demodulator in compressive sensing and make it work at a higher frequency. Herein, we first created an ideal Random Demodulator design in Matlab. In this design we use Gold Code to modulate input signals. The ideal CSFE Matlab design is shown in the figure 4.1 where waveforms at every stage is also displayed. In figure 4.1 a Gold code sequence of 127 bits is used as a repetitive bit sequence. A White Gaussian noise is added with an input signal at 5dB. The input is then multiplied with the repeated bit sequence by the 1-bit multiplier. If the Gold code bit is '1' then input is directly passed to the output and if the Gold code bit is '-1' then the 180 degree phase shift of input is passed to the output. Once multiplication is done, the output of multiplier is sent to the sample and hold circuit. We use discrete time integrator to function as sample and hold. It first samples the signal at the Gold code frequency and then feed the output to the integrator. Integrator functions as a low pass filter. Integrator is reset after every certain time which depends on the under sampling rate of ADC. The output of integrator is then digitized by an under-sampling ADC.



4.1 Ideal CSFE in Simulink (Matlab)

4.2 Orthogonal Matching Pursuit (OMP) algorithm for reconstruction

OMP is a greedy algorithm which is used for sparse signal reconstruction after compressive sensing front-end. We use OMP to rebuild the original signal from the under sampled data in CSFE. Assume our measurement matrix is Φ of a size $M \times N(M < N)$, and Y is a vector (size M) that is multiplication of input signal X (size N) and the measurement matrix Φ. Our aim using OMP is to get coefficient vector X or similar to X so that $\Phi \times$ X equals or approximately equals to Y. The OMP algorithm is presented in Algorithm 1 [16]. OMP has good approximation performance. Hypothetical examination of OMP to date has been analyzed basically on two fronts. The first involves coherent parameter $\boldsymbol{\mu} := \boldsymbol{max_{ij}}|\langle \Phi_i, \Phi_j \rangle$, Φ_i is column *I* of the matrix Φ. So whenever column *I* has the unit norm and parameter $\boldsymbol{\mu} < \frac{\mathbf{1}}{\mathbf{2K-1}}$, it has been shown that OMP is able to recover K- sparse signal X from the Y measurements [16]. The second thing involves the notion of probability. Let's assume that $X \in R^N$ with$\|X\|_0 := \boldsymbol{supp}(\boldsymbol{X}) \leq \boldsymbol{K}$ and Φ is randomly distributed and independent of X with $M = O(K \log(N))$ rows, then X can be recovered using OMP with very high probability.

Algorithm 1: Orthogonal Matching Pursuit **[16]**

Input: Φ, Y, atopping criterion

Initialize: $r^0 = \boldsymbol{Y}$, $\boldsymbol{X^0} = \mathbf{0}$, $\Lambda^0 = \emptyset$, $\boldsymbol{l} = \mathbf{0}$

While not converged **do**

Match: $\boldsymbol{h^l} = \Phi^T \boldsymbol{r^l}$

Identify: $\Lambda^{l+1} = \Lambda^l \cup \left\{\boldsymbol{argmax_j} \middle| \boldsymbol{h^l(j)}\right\}$ {if multiple maxima exists, choose only one}

Update: $x^{l+1} = \boldsymbol{argmin}_{\boldsymbol{z:supp(z)} \subseteq \Lambda^{l+1}} \|\boldsymbol{Y} - \Phi z\|_{\boldsymbol{2}}$

$\boldsymbol{r^{l+1}} = \boldsymbol{y} - \Phi x^{l+1}$

$l = l + 1$

End while

Output: $\hat{X} = \boldsymbol{x^{l}} = \boldsymbol{argmin}_{\boldsymbol{z:supp(z)} \subseteq \Lambda^{l+1}} \|\boldsymbol{Y} - \Phi z\|_{\boldsymbol{2}}$**.**

4.3 Proposed CSFE using Cadence and Matlab:

Compressed sensing front-end has four mian primary blocks. They are pseudo bit random sequence generator, multiplier, Integrator and ADC. As discussed earlier we have designed Pseudo random sequence generator (which is Gold Code in our case) and Multiplier in cadence using TSMC 180nm technology. Their Cadence designs were presented in chapter 3. Other primary blocks are implemented in Simulink. We integrate Cadence and Simulink for this work. To get the under sample data, we first give input signal to the multiplier. The input signal is a sine wave with White Gaussian noise added. We have tried many cases of different input signal strengths with different noise power to verify the sensitivity of the proposed design. Figure 4.2 shows the Gold code generator and 1-bit multiplier. The Gold code generator contains two 7 stages linear feedback shift registers, which generates 127 $(2^7 - 1)$ bits of Gold code sequence. It repeats the same cycle for every 127 bits.

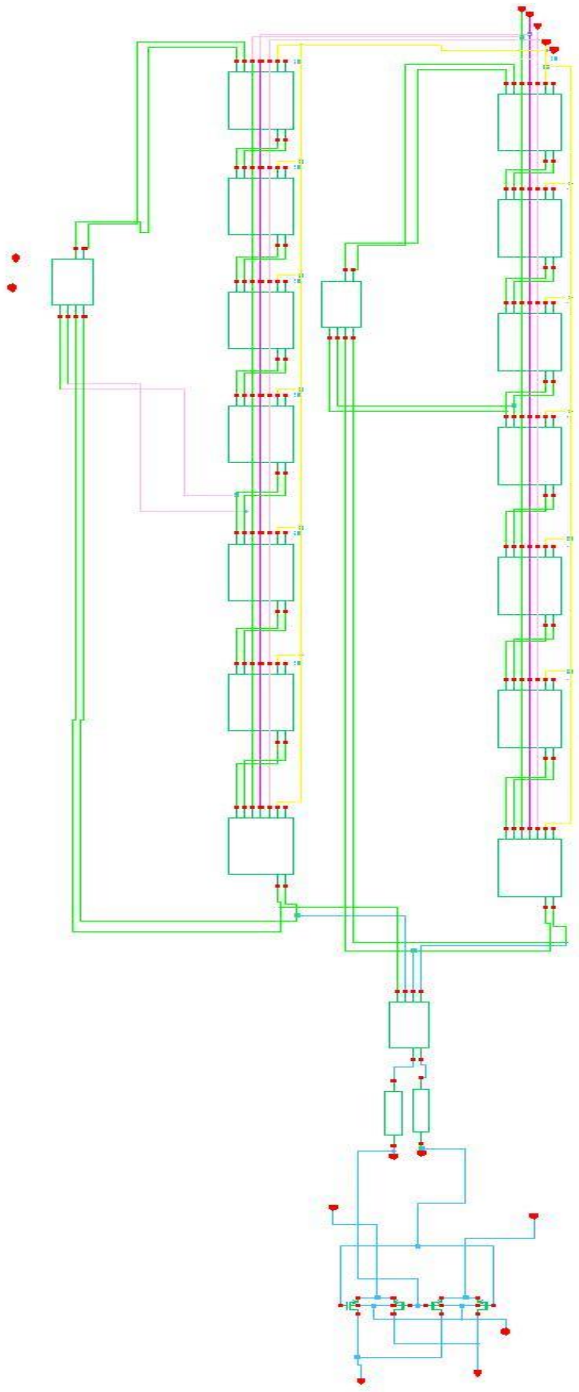
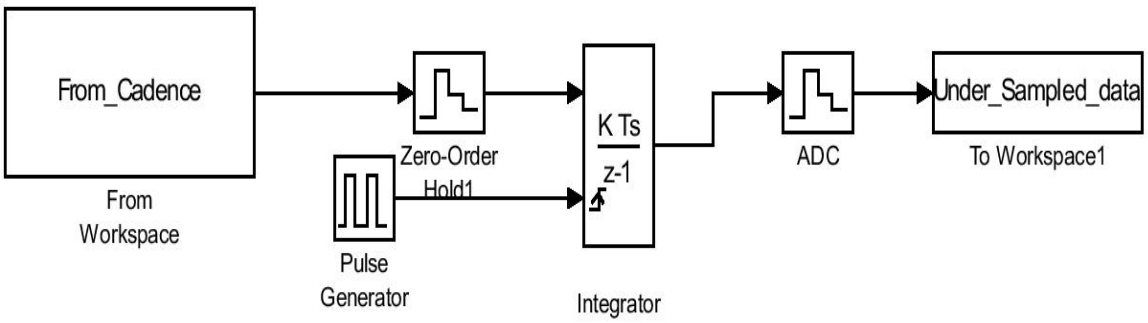


Fig. 4.2 Gold Code generator with Multiplier in Cadence



4.3 CSFE using Cadence and Simulink blocks

Figure 4.3 displays the final design of CSFE where the first block is the output of the cadence design. Other blocks are integrator and ADC in Simulink. The output of figure 4.3 is under sample data for signal reconstruction using Orthogonal Matching Pursuit (OMP). The measurement matrix Φ is built according to the Gold code. One example of how to construct measurement matrix is given below. Assume the Gold code sequence length is 7 bits which are: [1 -1 -1 1 1 -1 -1] and the under sampling rate is 1/3 then measurement matrix Φ is:

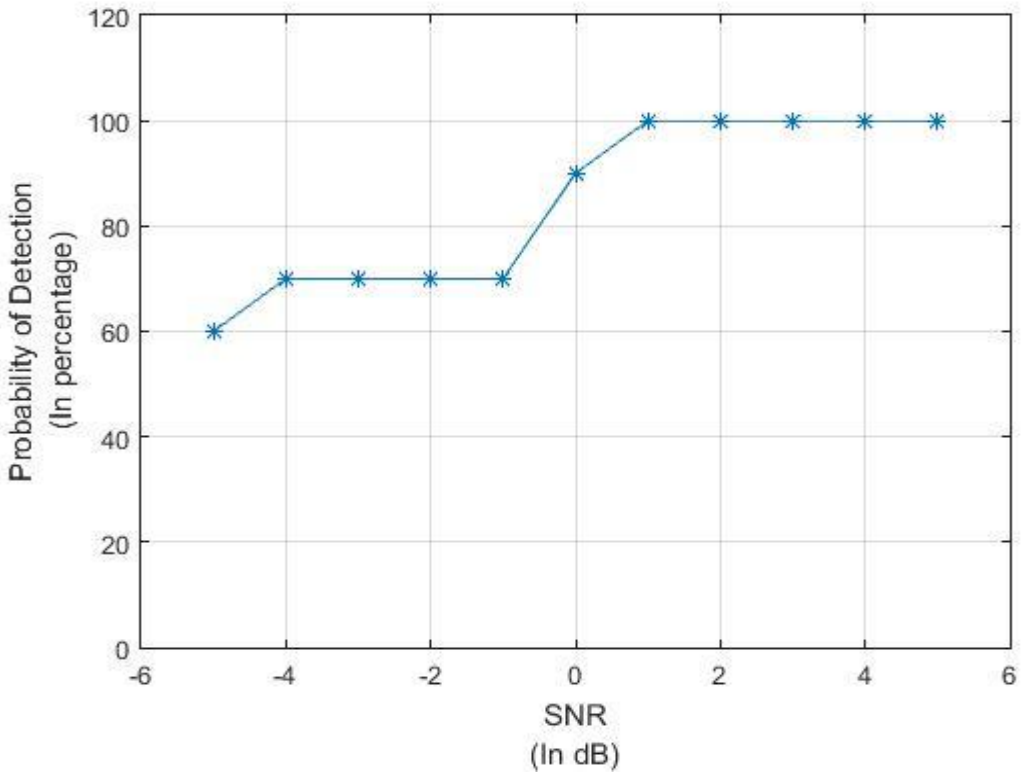
$$\begin{bmatrix} 1 & -1 & -1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & -1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & -1 & 1 & -1 \end{bmatrix}$$

Here, the measurement matrix size is 3×9. Here size of the measurement matrix is decided according to the under sampling rate. If the measurement matrix size is $M \times N$ then the under

sampling rate is M/N. In our first experiment the measurement size is 150 × 750. The under sampling rate is 5.

4.4 Sensitivity curve for 4 GHz sampling rate:

The proposed CSFE design and OMP successfully function at 4 GHz sampling rate. The sensitivity curve at 4GHz is shown in figure 4.4. The X-axes is SNR of input signal and Y-axes is probability of signal detection after OMP. For very case of SNR (from -5 to +5 dB) we randomly generate 10 signal frequency cases. The probability of signal detection is 100% at SNR from +1 to +5 dB, 90% at SNR of 0 dB, 70% at SNR from -1 to -4 dB, and 60% at SNR of -4 dB. The receiver sensitivity is 0 dB.



4.4 Sensitivity curve for 4 GHz

Chapter 5

Conclusion and Future work

5.1 Conclusion

In this research, we have investigated the new field of compressed sensing and proposed a front-end design for CS. We have proposed architecture design of CSFE, component designs using Cadence virtuoso in CMOS 180nm technology and Cadence using Verilog-A. We also built a CSFE design in Simulink to verify the performance of the proposed design. We have proposed Gold code generator in cadence, which can run up to 5 GHz. The Gold code generator is used as a pseudo random sequence generator. The length of the gold code sequence is also an important factor in the compressed sensing. The proposed design did not work for the 31 bits repetitive sequence but it worked perfectly fine for the 127 bits of repetitive sequence. A CMOS transistor design of multiplier has been presented in cadence. Ideal designs of integrator and ADC are presented in both Cadence Verilog-A and Simulink. We use OMP algorithm to reconstruct the under sample data. The new proposed design is able to reconstruct Nyquist data at 5 times under sampling rate of the Nyquist rate. The proposed design operates at 4 GHz and the ADC operates at an under sampling frequency of 800MHz, five times lower than the Nyquist rate. We have also evaluated the performance of CSFE and OMP for different input signal SNR. The receiver sensitivity is 0 dB.

5.2 Future work:

Future work includes:

- Designing new schematic level design of integrator and ADC with low power consumption and high speed.
- Designing a higher speed Gold code generator, which can run at frequency higher than 5GHz.
- Testing robustness of the CSFE design by adopting CMOS process variation and Monte Carlo analysis.

References

[1] Stephen Becker, “Practical compressed sensing: modern data acquisition and signal processing”, thesis, Caltech, 2011.

[2] J. Yoo, S. Becker, M. Loh, M. Monge, E. Candès, A. Emami-Neyestanak, “A 100MHz-2GHz 12.5x sub-Nyquist Rate Receiver in 90nm CMOS”, 2012 IEEE Radio Frequency Integrated Circuits Symposium (RFIC), Montreal, Canada.

[3] J. Yoo, S. Becker, M. Monge, M. Loh, E. Candès, A. Emami-Neyestanak, “Design and implementation of a fully integrated compressed-sensing signal acquisition system”, ICASSP 2012. (Kyoto, Japan, March 2012)

[4] J. Yoo, C. Turnes, E. Nakamura, C. Le, S. Becker, E. Sovero, M. Wakin, M. Grant, J. Romberg, A. Emami-Neyestanak, and E. Candès, “A Compressed Sensing Parameter Extraction Platform for Radar Pulse Signal Acquisition”. IEEE Journal on Emerging and Selected Topics in Circuits and Systems, 2012.

[5] E. J. Candès and M. B. Wakin, “An introduction to compressive sampling”, in IEEE Signal Processing Magazine, vol. 25, no. 2, pp. 21--30, March 2008.

[6] J. A. Tropp, J. N. Laska, M. F. Duarte, J. K. Romberg, and R. G. Baraniuk, “Beyond Nyquist: efficient sampling of sparse bandlimited signals”, IEEE Transactions on Information Theory, 2010.

[7] Sami Kirolos, Jason Laska, Michael Wakin, Marco Duarte, Dror Baron, Tamer Ragheb, Yehia Massoud, and Richard Baraniuk, “Analog-to-information conversion via random demodulation”. (IEEE Dallas Circuits and Systems Workshop (DCAS), Dallas, Texas, 2006)

[8] Jason Laska, Sami Kirolos, Marco Duarte, Tamer Ragheb, Richard Baraniuk, and Yehia Massoud, “Theory and implementation of an analog-to-information converter using random demodulation”. (IEEE Int. Symp. on Circuits and Systems (ISCAS), New Orleans, Louisiana, 2007)

[9] Tamer Ragheb, Jason N. Laska, Hamid Nejati, Sami Kirolos, Richard G. Baraniuk, and Yehia MassoudCwe, “A Prototype Hardware for Random Demodulation Based Compressive Analog-to-Digital Conversion”.

[10] P. Heydari and R. Mohanavelu, “Design of Ultrahigh-Speed Low-Voltage CMOS CML Buffers and Latches”, IEEE Transactions on Very Large Scale Integration (VLSI) System, vol. 12, no. 10, pp.1081-1093, Oct. 2004.

[11] M. George, M. Hamid, and A. Miller, "Gold Code Generators in Virtex Devices", Xilinx application note, XAPP217 (v1.1), Jan. 2001.

[12] C. -L. L, H. -C. Wang, 2, J. -C. Juang and H. -R. Chuang, “10-Gb/s CMOS Ultrahigh-Speed Gold-Code Generator Using Differential-Switches Feedback”.

[13] Richard Baraniuk, “Compressive Sensing”. [Lecture Notes in IEEE Signal Processing Magazine] Volume 24, July 2007

[14] Fred Chen, Anantha P. Chandrakasan and Vladimir M. Stojanović, “Design and Analysis of a Hardware-Efficient Compressed Sensing Architecture for Data Compression in Wireless Sensors”.

[15] “Nyquist-shannon sampling theorem”, Wikipedia 2011.

[16] Mark A. Davenport, Member, IEEE, and Michael B. Wakin, Member, “Analysis of Orthogonal Matching Pursuit Using the Restricted Isometry Property”.

Appendix

VHDL code of the Gold Code generator:

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
entity gd2 is
generic(n:integer:=5) ;
    Port ( clk : in  STD_LOGIC;
           reset : in  STD_LOGIC;
           g1 : inout  STD_LOGIC;
           g2 : inout  STD_LOGIC;
           p1 : out  STD_LOGIC);
end gd2;
architecture Behavioral of gd2 is
signal x1:std_logic_vector(0 to n-1);
signal x2:std_logic_vector(0 to n-1);
begin
P3:process(clk,reset)
begin
if(reset='1') then
x1<=(others=>'1');
G1<='0';
elsif(clk'event and clk='1') then
for m in 0 to 31 loop
  for i in 0 to n-2 loop
  x1 (i+1) <= x1(i);
x1(0) <= x1(2) xor x1(4);
  G1<=x1(4);
  end loop;
end loop;
end if ;
end process P3 ;
------------------  polynomial  G2 ----------------------
P2:process(reset,clk)
begin
if(reset='1') then
x2<=(others=>'1');
G2<='0';
elsif(clk'event and clk='1') then
for w in 0 to 31 loop
for j in 0 to n-2 loop
x2(j+1)<=x2(j);
end loop;
x2(0) <= ((x2(1) xor (x2(4))));
G2<=x2(4) ;
end loop;
end if ;
```

```
end process P2 ;
------------------ G1 xor G2 -------------
p1 <= G1 xor G2;
end Behavioral;
```

Verilog-A code for 10-bit ADC:

```
`include "discipline.h"
`include "constants.h"
`define NUM_ADC_BITS 10
module adc_8bit (vin, clk, data);
input vin, clk;
electrical vin, clk;
output [`NUM_ADC_BITS-1:0] data;
electrical [`NUM_ADC_BITS-1:0] data;
parameter real vmax = 0.25;
parameter real vmin = -0.25;
parameter real one = 1.8;
parameter real zero = 0;
parameter real vth = 0;
parameter real slack = 0.5p from (0:inf);
parameter real trise = 1.0p from (0:inf);
parameter real tfall = 1.0p from (0:inf);
parameter real tconv = 0.5p from [0:inf);
parameter integer traceflag = 1;
real sample, vref, lsb, voffset;
real vd[0:`NUM_ADC_BITS-1];
integer ii, binvalue;
analog begin
@(initial_step or initial_step("dc", "ac", "tran", "xf")) begin
vref = (vmax - vmin) / 2.0;
lsb = (vmax - vmin) / (1 << `NUM_ADC_BITS) ;
voffset = vmin;
if (traceflag)
$display("%M ADC range ( %g v ) / %d bits = lsb %g volts.\n",
vmax - vmin, `NUM_ADC_BITS, lsb );
generate i ( `NUM_ADC_BITS-1, 0) begin
vd[i] = 0 ;
end
end
@(cross ( V(clk)-vth, 1, slack, clk.potential.abstol)) begin
binvalue = 0;
sample = V(vin) - voffset;
for ( ii = `NUM_ADC_BITS -1 ; ii>=0 ; ii = ii -1 ) begin
vd[ii] = 0;
if (sample > vref ) begin
vd[ii] = one;
sample = sample - vref;
```

```
binvalue = binvalue + ( 1 << ii );
end
else begin
vd[ii] = zero;
end
sample = sample * 2.0;
end
if (traceflag)
$strobe("%M at %g sec. digital out: %d vin: %g (d2a: %g)\n",
$abstime, binvalue, V(vin), (binvalue*lsb)+voffset);
end
generate i ( `NUM_ADC_BITS-1, 0) begin
V(data[i]) <+ transition ( vd[i] , tconv, trise, tfall );
end
end
endmodule
`undef NUM_ADC_BITS
```

Verilog-A Code for an Integrator:

```
`include "discipline.h"
`include "constants.h"

// $Date: 1997/08/28 05:49:00 $
// $Revision: 1.1 $
//
//
// Based on the OVI Verilog-A Language Reference Manual, version 1.0
1996
//
//



//--------------------
// integrator
//
// -  integrator
//
// sigin:     (val,flow)
// sigout:    (val,flow)
//
// INSTANCE parameters
//    sigout0 = initial sigout value (val)
//    gain    = []
//
// MODEL parameters
//    {none}
//
```

```
module integrator(sigin, sigout);
input sigin;
output sigout;
electrical sigin, sigout;
parameter real sigout0 = 0;
parameter real gain = 1;

   analog
        V(sigout) <+ gain*idt(V(sigin), 0) + sigout0;

endmodule
```